全国计算机技术与软件专业技术资格（水平）考试用书

软件设计师考试

辅导教程

希赛教育软考学院　编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书由希赛教育软考学院组织编写，作为软件设计师考试辅导的指定教材。全书内容涵盖了考试大纲规定的所有知识点，对考试大纲规定的内容有重点地进行了细化和深化。阅读本书，就相当于阅读了一本详细的、带有知识注释的考试大纲。准备考试的人员可通过阅读本书掌握考试大纲规定的知识，掌握考试的重点和难点，熟悉内容的分布。

图书在版编目（CIP）数据

软件设计师考试辅导教程 / 希赛教育软考学院编著. —北京：电子工业出版社，2015.3
全国计算机技术与软件专业技术资格（水平）考试用书
ISBN 978-7-121-25614-1

Ⅰ. ①软… Ⅱ. ①希… Ⅲ. ①软件设计－工程技术人员－资格考试－自学参考资料 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2015）第 041244 号

策划编辑：孙学瑛
责任编辑：徐津平
特约编辑：赵树刚
印　　刷：
装　　订：
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×1092　1/16　印张：26　字数：749 千字
版　　次：2015 年 3 月第 1 版
印　　次：2015 年 3 月第 1 次印刷
定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

前　　言

全国计算机技术与软件专业技术资格（水平）考试是由国家人力资源和社会保障部、工业和信息化部组织和领导的国家级考试，具有很高的权威性，但这同时也决定了其考试范围的广度和深度都比较大，使许多考生在复习和准备上都遇到了很多的难题。为帮助广大考生顺利通过考试，希赛教育软考学院组织编写了本书。

内容超值，针对性强

由于考试大纲规定的考试知识点体系庞大，对考生而言，要学习的内容很多。为此，希赛教育软考学院组织有关专家对考试大纲进行了深入的分析，在此基础上编写了本书，以作为计算机技术与软件专业技术资格（水平）考试中的软件设计师级别的考试辅导指定教材。

本书根据软件设计师的考试大纲编写而成，内容紧扣大纲，全面实用。本书在组织和写作上，倾注了作者们的许多精力和心血。相信本书能够对考生提高通过率、有效地完成“考试过关”提供帮助。考生可通过阅读本书，迅速掌握考试所涉及的知识点，全面进行梳理和系统学习考试大纲中的内容。

作者权威，阵容强大

希赛教育（www.educity.cn）专业从事人才培养、教育产品开发和教育图书出版，在职业教育方面具有极高的权威性。特别是在在线教育方面稳居国内首位，其远程教育模式得到了国家教育部门的认可和推广。

希赛教育软考学院（www.educity.cn/rk）是全国计算机技术与软件专业技术资格（水平）考试的顶级培训机构，拥有近 20 名资深软考辅导专家，负责考试大纲制定及软考辅导教材的编写工作。近年来共组织编写和出版了 100 多本软考教材，内容涵盖初级、中级和高级的各个专业，包括教程系列、辅导系列、考点分析系列、冲刺系列、串讲系列、试题精解系列、疑难解答系列、全程指导系列、案例分析系列、指定参考用书系列及一本通 11 个系列。希赛教育软考学院的专家录制了软考培训视频教程、串讲视频教程、试题讲解视频教程和专题讲解视频教程 4 个系列的软考视频。其软考教材、软考视频和软考辅导为考生助考并提高通过率做出了不可磨灭的贡献，在软考领域有口皆碑。特别是在高级资格领域，无论是考试教材，还是在线辅导和面授，希赛教育软考学院都独占鳌头。

本书由希赛教育软考学院的王勇主编，参加编写工作的还有张友生、谢顺、刘洋波、桂阳、胡光超、邓旭光、左水林、胡钊源、王军、王玉罡。

在线测试，心中有数

希赛网在线测试平台（www.educity.cn/tiku/）为考生准备了在线测试，其中有数十套全真模拟试题和考前密卷，考生可选择任何一套进行测试。测试完毕，系统自动判卷，立即给出分数。

对于考生做错的地方，系统会自动记忆，待考生第二次参加测试时可选择“试题复习”。这样系统就会自动显示考生原来做错的试题，供重新测试，以加强记忆。

考生可利用希赛网在线测试平台的在线测试系统检查自己的实际水平，加强考前训练，做到心中有数，考试不慌。

诸多帮助，诚挚致谢

在本书的编写过程中参考了许多相关的文献和书籍，编者在此对这些参考文献的作者表示感谢。

感谢电子工业出版社的孙学瑛老师，她在本书的策划、选题的申报、写作大纲的确定，以及编辑和出版等方面付出了辛勤的劳动和智慧，给予我们很多的支持和帮助。

感谢参加希赛教育软考学院辅导和培训的学员，正是他们的想法汇成了本书的原动力，他们的意见使本书更加贴近读者。

由于编者水平有限且本书涉及的内容很广，所以书中难免存在错漏和不妥之处，编者诚恳地期望各位专家和读者不吝指正和帮助，对此我们将十分感激。

希赛教育软考学院
2015 年 2 月

目　录

第 20 章　常用算法设计......................378

第 1 章

数据结构基础

数据结构是指数据对象及其相互关系和构造方法，一个数据结构 S 可以用一个二元组表示为：S=（D，R）。其中，D 是数据结构中的数据的非空有限集合，R 是定义在 D 上的关系的非空有限集合。在数据结构中，结点及结点间的相互关系称为数据的逻辑结构，数据在计算机中的存储形式称为数据的存储结构。

数据结构按逻辑结构的不同分为线性结构和非线性结构两大类，其中非线性结构又可分为树形结构和图结构，而树形结构又可分为树结构和二叉树结构。

1.1　线性表

线性表是最简单、最常用的一种数据结构，它是由相同类型的结点组成的有限序列。一个由 n 个结点 a_0，a_1，…，a_{n-1} 组成的线性表可记为（a_0，a_1，…，a_{n-1}）。线性表的结点个数为线性表的长度，长度为 0 的线性表称为空表。对于非空线性表，a_0 是线性表的第一个结点，a_{n-1} 是线性表的最后一个结点。线性表的结点构成一个序列，对序列中两相邻结点 a_i 和 a_{i+1}，称 a_i 是 a_{i+1} 的前驱结点，a_{i+1} 是 a_i 的后继结点。其中 a_0 没有前驱结点，a_{n-1} 没有后继结点。

线性表中结点之间的关系可由结点在线性表中的位置确定，通常用（a_i，a_{i+1}）（$0 \leqslant i \leqslant n-2$）表示两个结点之间的先后关系。例如，如果两个线性表有相同的数据结点，但它们的结点在线性表中出现的顺序不同，则它们是两个不同的线性表。

线性表的结点可由若干成分组成，其中能唯一标识该结点的成分称为关键字，或简称键。为了讨论方便，往往只考虑结点的关键字，而忽略其他成分。

1．线性表的基本运算

线性表包含的结点个数可以动态增加或减少，可以在任何位置插入或删除结点。线性表常用的运算可分成几类，每类有若干种运算。

1）查找运算

在线性表中查找具有给定键值的结点。

2）插入运算

在线性表的第 i（$0 \leqslant i \leqslant n-1$）个结点的前面或后面插入一个新结点。

3）删除运算

删除线性表的第 i（$0 \leqslant i \leqslant n-1$）个结点。

4）其他运算

- 统计线性表中结点的个数。
- 输出线性表各结点的值。
- 复制线性表。
- 线性表分拆。
- 线性表合并。
- 线性表排序。
- 按某种规则整理线性表。

2．线性表的存储

线性表常用的存储方式有顺序存储和链接存储。

1）顺序存储

顺序存储是最简单的存储方式，通常用一个数组，从数组的第一个元素开始，将线性表的结点依次存储在数组中，即线性表的第 i 个结点存储在数组的第 i（$0 \leqslant i \leqslant n-1$）个元素中，用数组元素的顺序存储来体现线性表中结点的先后次序关系。

顺序存储线性表的最大优点就是能随机存取线性表中的任何一个结点，缺点主要有两个，一是数组的大小通常是固定的，不利于任意增加或减少线性表的结点个数；二是插入和删除线性表的结点时，要移动数组中的其他元素，操作复杂。

2）链接存储

链接存储是用链表存储线性表（链表），最简单的是用单向链表，即从链表的第一个结点开始，将线性表的结点依次存储在链表的各结点中。链表的每个结点不但要存储线性表结点的信息，还要用一个域存储其后继结点的指针。单向链表通过链接指针来体现线性表中结点的先后次序关系。

链表存储线性表的优点是线性表中每个结点的实际存储位置是任意的，这给线性表的插入和删除操作带来了方便，只要改变链表有关结点的后继指针就能完成插入或删除的操作，不需移动任何表元。链表存储方式的缺点主要有两个，一是每个结点增加了一个后继指针成分，要花费更多的存储空间；二是不便随机访问线性表的任一结点。

3．线性表上的查找

线性表上的查找运算是指在线性表中找某个键值的结点。根据线性表中的存储形式和线性表本身的性质差异，有多种查找算法，如顺序查找、二分法查找、分块查找、散列查找等。其中二分法查找要求线性表是一个有序序列。

4．在线性表中插入新结点

1）顺序存储

设线性表结点的类型为整型，插入之前有 n 个结点，把值为 x 的新结点插在线性表的第 i（$0 \leqslant i \leqslant n$）个位置上。完成插入主要有如下步骤：

- 检查插入要求的有关参数的合理性。
- 把原来的第$n-1$个结点至第i个结点依次往后移一个数组元素位置。
- 把新结点放在第i个位置上。

- 修正线性表的结点个数。

在具有 n 个结点的线性表上插入新结点，其时间主要花费在移动结点的循环上。若插入任一位置的概率相等，则在顺序存储线性表中插入一个新结点，平均移动次数为（n-1）/2。

2）链接存储

在链接存储线性表中插入一个键值为 x 的新结点，分为如下 4 种情况：

- 在某指针 p 所指结点之后插入。
- 插在首结点之前，使待插入结点成为新的首结点。
- 接在线性表的末尾。
- 在有序链表中插入，使新的线性表仍然有序。

5．删除线性表的结点

1）顺序存储

在有 n 个结点的线性表中，删除第 i（$0 \leqslant i \leqslant n-1$）个结点。删除时应将第 i+1 个结点至第 $n-1$ 个结点依次向前移一个数组元素位置，共移动 $n-i-1$ 个结点。完成删除主要有如下几个步骤：

- 检查删除要求的有关参数的合理性。
- 把原来第 i+1 个表元至第 $n-1$ 个结点依次向前移一个数组元素位置。
- 修正线性表表元个数。

在具有 n 个结点的线性表上删除结点，其时间主要花费在移动表元的循环上。若删除任一表元的概率相等，则在顺序存储线性表中删除一个结点，平均移动次数为 n/2。

2）链接存储

对于链表上删除指定值结点的删除运算，需考虑几种情况，一是链表为空链表，不执行删除操作；二是要删除的结点恰为链表的首结点，应将链表头指针改为指向原首结点的后继结点；其他情况，先要在链表中寻找要删除的结点，从链表首结点开始顺序寻找。若找到，执行删除操作，若直至链表末尾没有指定值的结点，则不执行删除操作。完成删除由如下几个步骤组成：

- 如链表为空链表，则不执行删除操作。
- 若链表的首结点的值为指定值，更改链表的头指针为指向首结点的后继结点。
- 在链表中寻找指定值的结点。
- 将找到的结点删除。

1.1.1 栈

栈是一种特殊的线性表，栈只允许在同一端进行插入和删除运算。允许插入和删除的一端称为栈顶，另一端为栈底。称栈的结点插入为进栈，结点删除为出栈。因为最后进栈的结点必定最先出栈，所以栈具有后进先出的特征。

1．顺序存储

可以用顺序存储线性表来表示栈，为了指明当前执行插入和删除运算的栈顶位置，需要一个地址变量 top 指出栈顶结点在数组中的下标。

2．链接存储栈

栈也可以用链表实现，用链表实现的栈称为链接栈。链表的第一个结点为顶结点，链表的首结点就是栈顶指针 top，top 为 NULL 的链表是空栈。

1.1.2 队列

队列也是一种特殊的线性表，只允许在一端进行插入，另一端进行删除运算。允许删除运算的那一端称为队首，允许插入运算的一端称为队尾。称队列的结点插入为进队，结点删除为出队。因最先进入队列的结点将最先出队，所以队列具有先进先出的特征。

1．顺序存储

可以用顺序存储线性表来表示队列，为了指明当前执行出队运算的队首位置，需要一个指针变量 head（称为头指针），为了指明当前执行进队运算的队尾位置，也需要一个指针变量 tail（称为尾指针）。

若用有 N 个元素的数组表示队列，随着一系列进队和出队运算，队列的结点移向存放队列的数组的尾端，会出现数组的前端空着，而队列空间已用完的情况。一种可行的解决办法是当发生这样的情况时，把队列中的结点移到数组的前端，修改头指针和尾指针。另一种更好的解决办法是采用循环队列。

循环队列就是将实现队列的数组 $a[N]$的第一个元素 $a[0]$与最后一个元素 $a[N-1]$连接起来。队空的初态为 head=tail=0。在循环队列中，当 tail 赶上 head 时，队列满。反之，当 head 赶上 tail 时，队列变为空。这样队空和队满的条件都同为 head=tail，这会给程序判别队空或队满带来不便。因此，可采用当队列只剩下一个空闲结点的空间时，就认为队列已满的简单办法，以区别队空和队满。即队空的判别条件是 head=tail，队满的判别条件是 head=tail+1。

2．链接存储

队列也可以用链接存储线性表实现，用链表实现的队列称为链接队列。链表的第一个结点是队列首结点，链表的末尾结点是队列的队尾结点，队尾结点的链接指针值为 NULL。队列的头指针 head 指向链表的首结点，队列的尾指针 tail 指向链表的尾结点。当队列的头指针 head 值为 NULL 时，队列为空。

1.1.3 稀疏矩阵

在计算机中存储一个矩阵时，可使用二维数组。例如，$M \times N$ 阶矩阵可用一个数组 $a[M][N]$来存储（可按照行优先或列优先的顺序）。如果一个矩阵的元素绝大部分为零，则称为稀疏矩阵。若直接用一个二维数组表示稀疏矩阵，则会因存储太多的零元素而浪费大量的内存空间。因此，通常采用三元组数组或十字链表两种方法来存储稀疏矩阵。

1．三元组数组

稀疏矩阵的每个非零元素用一个三元组来表示，即非零元素的行号、列号和它的值。然后按某种顺序将全部非零元素的三元组存于一个数组中。

如果只对稀疏矩阵的某些单个元素进行处理，则宜用三元组表示。

2．十字链表

在十字链表中，矩阵的非零元素是一个结点，同一行的结点和同一列的结点分别顺序

循环链接，每个结点既在它所在行的循环链表中，又在它所在列的循环链表中。每个结点含 5 个域，分别为结点对应的矩阵元素的行号、列号、值，以及该结点所在行链表后继结点指针、所在列链表后继结点指针。

为了处理方便，通常对每个行链表和列链表分别设置一个表头结点，并使它们构成带表头结点的循环链表。为了引用某行某列的方便，全部行链表的表头结点和全部列链表的表头结点分别组成数组，这两个数组的首结点指针存于一个十字链表的头结点中，最后由一个指针指向该头结点。

例如，矩阵 $\boldsymbol{A}$ 如图 1-1 所示。

$$\begin{bmatrix} 5 & 9 & 2 \\ 3 & 0 & 0 \\ 8 & 0 & 0 \end{bmatrix}$$

图 1-1　矩阵 $\boldsymbol{A}$ 示意图

则其十字链表如图 1-2 所示。

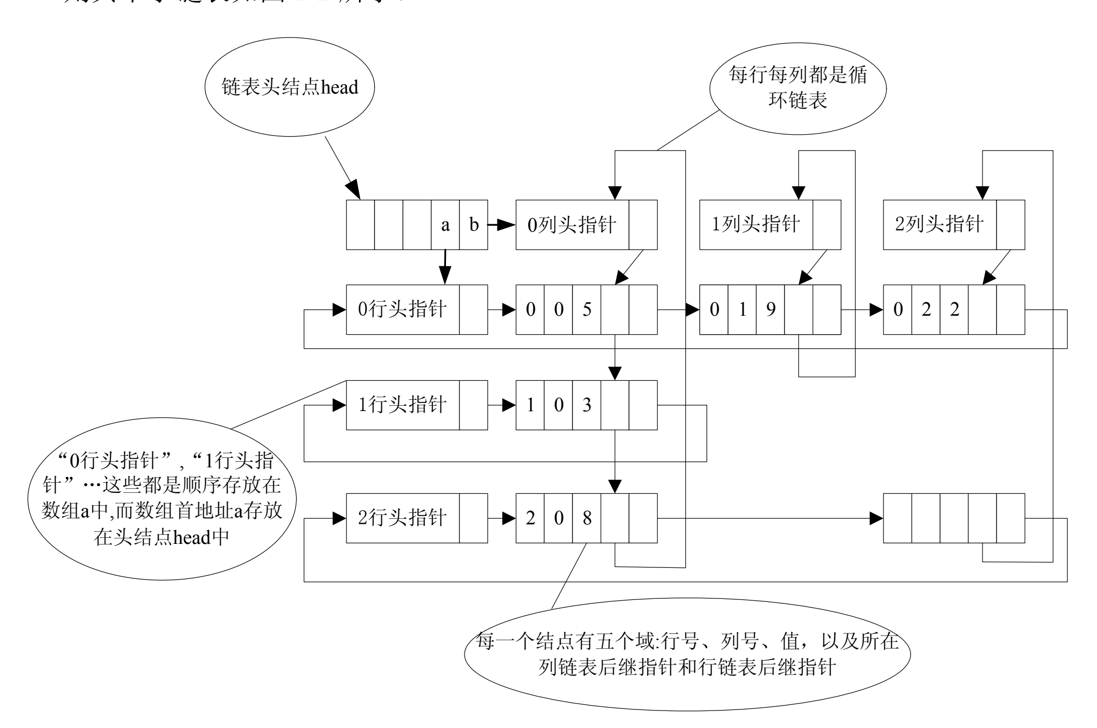


图 1-2　矩阵 $\boldsymbol{A}$ 十字链表存储示意图

如果对稀疏矩阵某行或某列整体做某种处理，可能会使原来为零的元素变为非零，而原来非零的元素变成零。对于这种场合，稀疏矩阵宜用十字链表来表示。

1.1.4　字符串

字符串是由某字符集上的字符所组成的任何有限字符序列。当一个字符串不包含任何字符时，称为空字符串。一个字符串所包含的有效字符个数称为这个字符串的长度。一个字符串中任一连续的子序列称为该字符串的子串。

1．字符串的存储

字符串通常存于足够大的字符数组中，每个字符串的最后一个有效字符之后有一个字符串结束标志，记为“\0”。通常由系统提供的库函数形成的字符串的末尾会自动添加“\0”，但当由用户的应用程序来形成字符串时，必须由程序自行负责在最后一个有效字符之后添加“\0”，以形成字符串。

2．字符串的操作

对字符串的操作通常有：

- 统计字符串中有效字符的个数。
- 把一个字符串的内容复制到另一个字符串中。
- 把一个字符串的内容连接到另一个足够大的字符串的末尾。
- 在一个字符串中查找另一个字符串（字符串匹配）或字符。
- 按字典顺序比较两个字符串的大小。

3．KMP 算法

KMP 算法是一种改进的字符串匹配算法，由 D.E.Knuth 与 V.R.Pratt 和 J.H.Morris 同时发现，因此人们称它为克努特——莫里斯——普拉特操作（简称 KMP 算法）。KMP 算法相比传统的字符串匹配算法 BF 算法，有更高的效率，但其处理过程并不是很好理解。在软件设计师考试中，经常考查该算法中 next 数组的内容分析。

为了更好地理解 KMP 算法，首先要理解 BF 算法的操作过程。若要匹配 S 串中，有没有存在 T 串。则 BF 算法的核心思想是：首先 $S[1]$和 $T[1]$比较，若相等，则再比较 $S[2]$和 $T[2]$，一直到 $T[M]$为止；若 $S[1]$和 $T[1]$不等，则 S 的下标变更加 1，再依次进行比较，即 $S[2]$和 $T[1]$比较。如果存在 k，使 $S[k+1 \cdots k+M]=T[1 \cdots M]$，则匹配成功；否则失败。该算法效率表现最差的一面在于：当比较了 $S[1]$与 $T[1]$、$S[2]$与 $T[2]$、$S[3]$与 $T[3]$、$S[4]$与 $T[4]$，它们都相等，而 $S[5]$与 $T[5]$不相等时，BF 算法的做法是将 S 数组的下标回退到 2，将 $S[2]$与 $T[1]$对比，然后依次类推，这样做是非常低效的。所以该算法最坏情况下要进行 $M*(N-M+1)$次比较，时间复杂度为 $O(M*N)$。

在 KMP 算法中，主要改进的，也就是 BF 算法的劣势。KMP 算法中会构造一个 next 数组，这个数组用于记录，当某个字符匹配不成功时，接下来应该进行哪个元素的对比。下面以一个实例分析 next 数组是如何计算出来的。

例如，在字符串的 KMP 模式匹配法中，需要求解模式串 p 的 next 函数值，其定义如下。若模式串 p 为“aaabaaa”，则其 next 函数值为______。

$$\text{next}[j]=\begin{cases}0 & j=1\\ \max\{k \mid 1<k<j, 'p_1p_2\ldots p_{k-1}'='p_{j-k+1}p_{j-k+2}\ldots p_{j-1}'\} \\ 1 & \text{其他情况}\end{cases}$$

A．0123123　　B．0123210　　C．0123432　　D．0123456

例题分析

KMP 模式匹配算法通俗来讲就是一种在一个字符串中定位另一个串的高效算法。其实在做这个题目时，也可以不需要知道 KMP 模式匹配算法，可以根据题目给出的定义式来求解。

当 j=1 时，很显然 next[1]=0。

当 j=2 时，由于 $1<k<j$，因此 k 无法取到合适值，因此 next[2]=1。

当 j=3 时，k 的取值为 2，那么等号左边的‘$P_1P_2\cdots P_{K-1}$’字符串就是 P_1，为字符串中的第一个字符 a，而右边就是 P_2，即字符串中的第二个字符 a。显然，它们相等。因此 next[3]=k=2。

当 j=4 时，k 可以取值 2 或者 3，取值为 2 时，等号左边为第一个字符 a，而等号右边为 P_3，也是字符 a，因此相等，但此时还要判定当 k 取值为 3 时，等号左边为第一个字符与第二个字符，即‘aa’，而右边为‘aa’，也相等。因此 next[4]=max{2,3}=3。

当 j=5 时，k 可以取值 2、3 或者 4，当 k 取值为 2 时，等号左边为第一个字符 a，而等号右边为 P_4，也是字符 b，不相等。当 k 取值为 3 时，等号左边为第一个字符与第二个字符，即‘aa’，而右边为‘ab’，也不相等。当 k 取值为 4 时，等号左边为“aaa”，而等号右边为“aab”，也不相等，因此 next[4]=1。

同理可以求得当 j=6，j=7 时的结果，本题正确答案选 A。

从上述分析可以看出，在 next 中记录的实际上是当前位置之前，已有多少个字符与串头相同。aaabaaa 中的第 3 个字母 a，对应的 next[3]=2，因为在这个字母之前，有两个字母都是 a，这与原串“aaabaaa”的前两个字母是相同的。而 next[6]=1，因为在第 6 个字母之前，有 1 个字母 a 与原串“aaabaaa”的第 1 个字母相同。这样，在进行字符串匹配过程中，如果匹配到第 6 个字母失配，则将第 6 个字母与模式串的 T[1]直接比较，而不用回退 S 串，其时间复杂度会大大降低。

1.2 树和二叉树

树结构是数据结构中的第二种经典结构，应用中主要使用的是二叉树，所以考试的重点在二叉树部分，本节也将重点描述二叉树的相关操作与特性。

1.2.1 树

1．树的基本概念

树是由一个或多个结点组成的有限集合 T，它满足如下两个条件：

（1）有一个特定的结点，称为根结点。

（2）其余的结点分成 m（$m\geqslant 0$）个互不相交的有限集合。其中每个集合又都是一棵树，称 T_1，T_2，…，T_{m-1} 为根结点的子树。

显然，上述定义是递归的，即一棵树由子树构成，子树又由更小的子树构成。由条件（1）可知，一棵树至少有一个结点（根结点）。一个结点的子树数目称为该结点的度（次数），树中各结点的度的最大值称为树的度（树的次数）。度为 0 的结点称为叶子结点（树叶），除叶子结点外的所有结点称为分支结点，根以外的分支结点称为内部结点。例如，在如图 1-3 所示的树中，根结点的度数为 3，结点 2 的度数为 4，结点 4 的度数为 1，结点 9 的度数为 2，其他结点的度数为 0，该树的度数为 4。

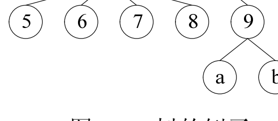


图 1-3　树的例子

在用图形表示的树中，对两个用线段连接的相关联的结点而言，称位于上端的结点是位于下端的结点的父结点或双亲结点，称位于下端的结点是位于上端的结点的（孩）子结点，称同一父结点的多个子结点为兄弟结点，称处于同一层次上、不同父结点的子结点为堂兄弟结点。例如，在图 1-3 中，结点 1 是结点 2、3、4 的父结点。反之，结点 2、3、4 都是结点 1 的子结点。结点 2、3、4 是兄弟结点，而结点 5、6、7、8、9 是堂兄弟结点。

定义一棵树的根结点所在的层次为 1，其他结点所在的层次等于它的父结点所在的层次加 1。树中各结点的层次的最大值称为树的层次。

2．树的常用存储结构

因为树是非线性的结构，为了存储树，必须要把树中结点之间的关系反映在存储结构中。最常用树的存储结构有标准存储和带逆存储形式。

1）标准存储结构

在树的标准存储结构中，树中的结点内容可分成两部分，分别为结点的数据和指向子结点的指针数组。对于 N 度树，在其标准存储结构中指针数组有 N 个元素。

例如，设树的度数为 5，树的结点数据仅限于字符，用 C 语言描述树结点的标准存储结构的数据类型如下：

```
#define N 5
typedef struct tnode{
char data; /*树结点的数据信息*/
struct tnode *child[N]; /*树结点的子结点指针*/
}TNODE; /*树结点的数据类型*/
```

2）带逆存储结构

带逆存储结构在标准存储结构的基础上增加一个指向其父结点的指针，用 C 语言描述树结点的带逆存储结构的数据类型如下：

```
#define N 5
typedef struct rtnode{
char data; /*树的结点数据信息*/
struct rtnode *child[N]; /*树结点的子结点指针*/
struct rtnode *parent; /*父结点指针*/
}RTNODE; /*树结点的数据类型*/
```

3．树的遍历

按照某种顺序逐个获得树中全部结点的信息，称为树的遍历。常用的树的遍历方法主要有如下 3 种。

- 前序遍历：首先访问根结点，然后从左到右按前序遍历根结点的各棵子树。
- 后序遍历：首先从左到右按后序遍历根结点的各棵子树，然后访问根结点。
- 层次遍历：首先访问处于0层上的根结点，然后从左到右依次访问处于1层上的结点，再从左到右依次访问处于2层上的结点等，即自上而下、从左到右逐层访问树中各层上的结点。

按上述遍历的定义，图 1-3 所示的树的各种遍历结果如下。

- 前序遍历：1，2，5，6，7，8，3，4，9，a，b。
- 后序遍历：5，6，7，8，2，3，a，b，9，4，1。
- 层次遍历：1，2，3，4，5，6，7，8，9，a，b。

1.2.2　二叉树

1．二叉树的基本概念

二叉树是一个有限的结点集合，该集合或者为空，或者由一个根结点及其两棵互不相交的左、右二叉子树所组成。二叉树的结点中有两棵子二叉树，分别称为左子树和右子树。因为二叉树可以为空，所以二叉树中的结点可能没有子结点，也可能只有一个左子结点（右子结点），也可能同时有左、右两个子结点。图 1-4 所示为二叉树的 4 种不同形态（如果把空树计算在内，则共有 5 种形态）。

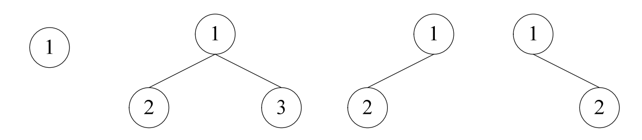


图 1-4　二叉树的 4 种不同形态

与树相比，二叉树可以为空，空的二叉树没有结点（树至少有一个结点）。在二叉树中，结点的子树是有序的，分左、右两棵子二叉树。

二叉树常采用类似树的标准存储结构来存储，其结点类型可以用 C 语言定义如下：

```
typedef struct Btnode{
char data; /*数据*/
strucrt Btnode *lchild; /*左孩子*/
struct Btnode *rchild; /*右孩子*/
}BTNODE;
```

2．二叉树的性质

二叉树具有下列重要性质（此处省略了推导过程，有兴趣的读者可自行推导）。

性质 1：在二叉树的第 i 层上至多有 2^{i-1} 个结点（$i \geqslant 1$）。

性质 2：深度为 k 的二叉树至多有 2^k-1 个结点（$k \geqslant 1$）。

性质 3：对任何一棵二叉树，如果其叶子结点数为 n_0，度为 2 的结点数为 n_2，则 $n_0 = n_2 + 1$。

一棵深度为 k 且有 2^k-1（$k \geqslant 1$）个结点的二叉树称为满二叉树。图 1-5 所示为一棵满二叉树，对结点进行了顺序编号。

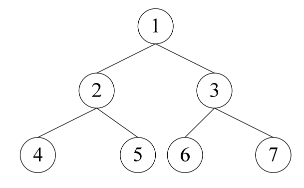


图 1-5　满二叉树的例子

如果深度为 k、有 n 个结点的二叉树中各结点能够与深度为 k 的顺序编号的满二叉树从 1 到 n 标号的结点相对应，则称这样的二叉树为完全二叉树。图 1-6（a）所示为一棵完全二叉树，而图 1-6（b）、图 1-6（c）所示为两棵非完全二叉树。显然，满二叉树是完全二叉树的特例。

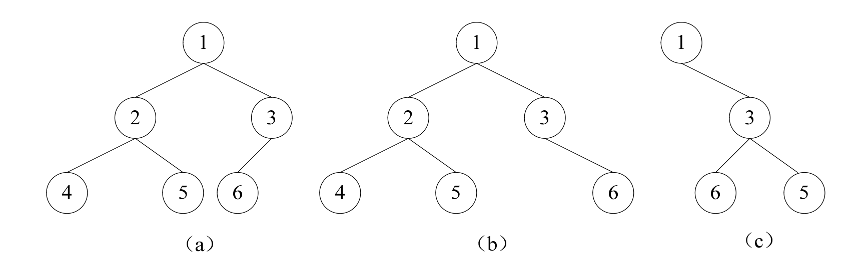


图 1-6　完全二叉树和非完全二叉树

根据完全二叉树的定义，显然，在一棵完全二叉树中，所有的叶子结点都出现在第 k 层或 $k-1$ 层（最后两层）。

性质 4：具有 n（$n>0$）个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor +1$（注：$\lfloor\ \rfloor$ 符号为向下取整运算符，$\lceil\ \rceil$ 为向上取整运算符，$\lfloor m \rfloor$ 表示不大于 m 的最大整数，反之，$\lceil m \rceil$ 表示不小于 x 的最小整数）。

性质 5：如果对一棵有 n 个结点的完全二叉树的结点按层序编号（从第 1 层到第 $\lfloor \log_2 n \rfloor +1$ 层，每层从左到右），则对任一结点 i（$1 \leqslant i \leqslant n$），有：

- 如果 $i=1$，则结点 i 无双亲，是二叉树的根；如果 $i>1$，则其双亲是结点 $\lfloor i/2 \rfloor$。
- 如果 $2i>n$，则结点 i 为叶子结点，无左孩子；否则，其左孩子是结点 $2i$。
- 如果 $2i+1>n$，则结点 i 无右孩子；否则，其右孩子是结点 $2i+1$。

3．二叉树的遍历

树的所有遍历方法也同样适用于二叉树，此外，由于二叉树自身的特点，还有中序遍历方法。

- 前序遍历（先根遍历、先序遍历）：首先访问根结点，然后按前序遍历根结点的左子树，再按前序遍历根结点的右子树。
- 中序遍历（中根遍历）：首先按中序遍历根结点的左子树，然后访问根结点，再按中序遍历根结点的右子树。
- 后序遍历（后根遍历、后序遍历）：首先按后序遍历根结点的左子树，然后按后序遍历根结点的右子树，再访问根结点。

例如，如图 1-7 所示的二叉树，其前序遍历、中序遍历和后序遍历结果分别如下。

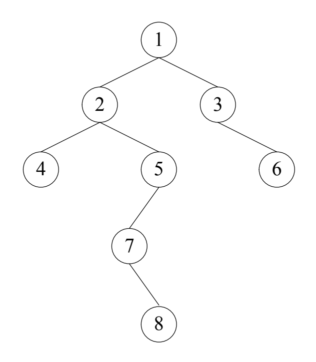


图 1-7　二叉树遍历的例子

• 前序遍历：1，2，4，5，7，8，3，6。
• 中序遍历：4，2，7，8，5，1，3，6。
• 后序遍历：4，8，7，5，2，6，3，1。

上述 3 种遍历方法都是递归定义的，可通过递归函数分别加以实现。

性质 6：一棵二叉树的前序序列和中序序列可以唯一地确定这棵二叉树。

根据性质 6，给定一棵二叉树的前序序列和中序序列，可以写出该二叉树的后序序列。例如，某二叉树的前序序列为 ABHFDECKG，中序序列为 HBDFAEKCG，则构造二叉树的过程如图 1-8 所示。

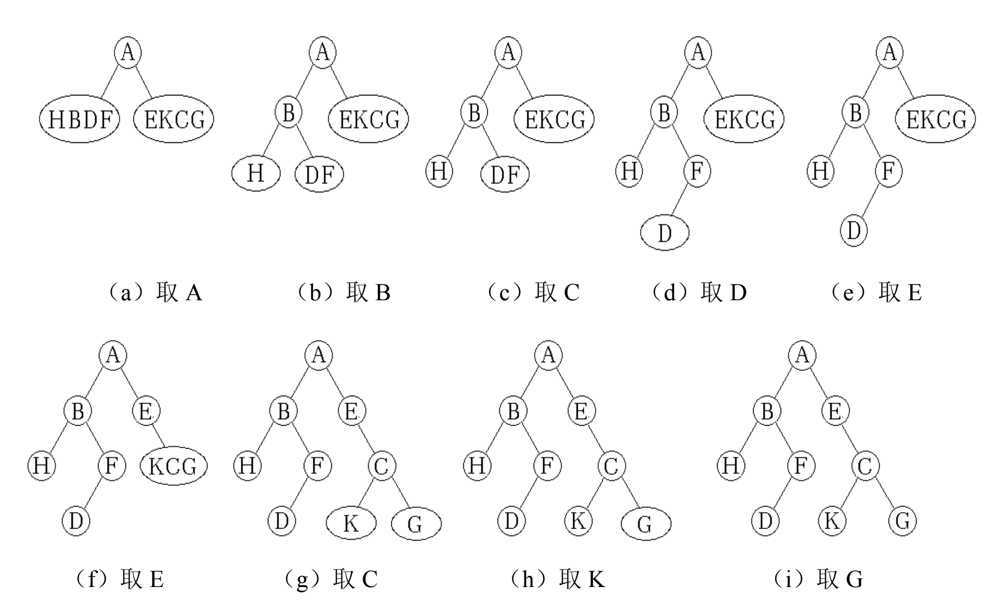


图 1-8　已知前序序列和中序序列，求二叉树的过程

1.2.3　二叉排序树

二叉排序树又称为二叉查找树，其定义为二叉排序树或者一棵空二叉树，或者具有如下性质（BST 性质）的二叉树：

（1）若它的左子树非空，则左子树上所有结点的值均小于根结点。

（2）若它的右子树非空，则右子树上所有结点的值均大于根结点。

（3）左、右子树本身又各是一棵二叉排序树。

例如，如图 1-9 所示为一棵二叉排序树。

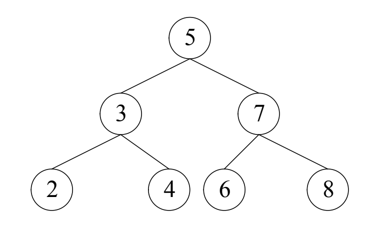


图 1-9　二叉排序树的例子

根据二叉排序树的定义可知，如果中序遍历二叉排序树，就能得到一个排好序的结点序列。二叉排序树上有查找、插入和删除 3 种操作。下面假设二叉排序树的结点只存储结点的键值，其类型与前面的二叉树的结点类型相同。

1．静态查找

静态查找是在二叉排序树上查找键值为 key 的结点是否存在，可按如下步骤在二叉排序树 ST 上查找值为 key 的结点：

- 如果二叉排序树ST为空二叉树，则查找失败，结束查找。
- 如果二叉排序树的根结点的键值等于key，则查找成功，结束查找。
- 如果key小于根结点的键值，则沿着根结点的左子树查找，即将根结点的左子树作为新的二叉排序树ST继续查找。
- 如果key大于根结点的键值，则沿着根结点的右子树查找，即将根结点的右子树作为新的二叉排序树ST继续查找。

2．动态查找

在二叉排序树上，为插入和删除操作需要而使用的查找称为动态查找。动态查找应得到两个指针，一个指向键值为 key 的结点，另一个指向该结点的父结点。为此，查找函数可设 4 个参数，查找树的根结点指针 root，待查找值 key，存储键值为 key 结点的父结点的指针 pre，存储键值为 key 结点的指针 p，但函数要考虑如下几种不同情况。

- 二叉排序树为空，查找失败，函数使*p=NULL，*pre=NULL。
- 二叉排序树中没有键值为key的结点，函数一直寻找至查找路径的最后一个结点，*pre指向该结点，*p=NULL，如果插入键值为key的结点，就插在该结点下。
- 查找成功，*p指向键值为key的结点，*p指向它的父结点。

3．插入结点

将利用动态查找函数确定新结点的插入位置，然后分如下几种情况进行相应的处理。

- 如果相同键值的结点已在二叉排序树中，则不再插入。
- 如果二叉排序树为空树，则以新结点为二叉排序树。
- 将要插入结点的键值与插入后的父结点的键值比较，就能确定新结点是父结点的左子结点，还是右子结点，并进行相应插入。

4．删除结点

删除二叉排序树上键值为 key 的结点的操作如下。

- 调用查找函数确定被删结点的位置。
- 如被删结点不在二叉排序树上，则函数返回。

否则，按如下情况分别处理。

- 如果被删除的结点是根结点，又可分两种情况：
  - 被删除结点无左子树，则以被删除结点的右子树作为删除后的二叉排序树。
  - 被删除结点有左子树，则以被删除结点的左子结点作为根结点，并把被删除结点的右子树作为被删除结点的左子树按中序遍历的最后一个结点的右子树。
- 如果被删除结点不是根结点，且被删除结点无左子结点，则分为如下两种情况：
  - 被删除结点是它的父结点的左子结点，则把被删除结点的右子树作为被删除结点的父结点的左子树。
  - 被删除结点是它的父结点的右子结点，则把被删除结点的右子树作为被删除结点的父结点的右子树。

- 如果被删除结点不是根结点，且被删除结点有左子结点，则被删除结点的右子树作为被删除结点的左子树按中序遍历的最后一个结点的右子树，同时进行如下操作：
  - 被删除结点是它的父结点的左子结点，则把被删除结点的左子树作为被删结点的父结点的左子树。
  - 被删除结点是它的父结点的右子结点，则把被删除结点的左子树作为被删除结点的父结点的右子树。

1.2.4　平衡二叉树

为了保证二叉排序树的高度为 $\log_2 n$，从而保证二叉排序树上实现的插入、删除和查找等基本操作的平均时间为 $O(\log_2 n)$，在往树中插入或删除结点时，要调整树的形态来保持树的“平衡”。使之既保持 BST 性质不变，又保证树的高度在任何情况下均为 $\log_2 n$，从而确保树上的基本操作在最坏情况下的时间均为 $O(\log_2 n)$。

平衡二叉树（Balanced Binary Tree 或 Height-Balanced Tree）又称为 AVL 树，是指树中任一结点的左、右子树的高度大致相同。如果任一结点的左、右子树的高度均相同（如满二叉树），则二叉树是完全平衡的。通常，只要二叉树的高度为 $O(1\text{og}_2 n)$，就可看作是平衡的。

平衡的二叉排序树指满足 BST 性质的平衡二叉树。AVL 树中任一结点的左、右子树的高度之差的绝对值不超过 1。若将二叉树上结点的平衡因子定义为该结点的左子树的深度减去它的右子树的深度，则平衡二叉树上所有结点的平衡因子只可能是－1、0 和 1。

在最坏情况下，n 个结点的 AVL 树的高度约为 $1.44\log_2 n$。而完全平衡的二叉树高度约为 $\log_2 n$，AVL 树接近最优。

1.2.5　线索树

二叉树在一般情况下无法直接找到某结点在某种遍历序列中的前驱和后继结点。若增加指针域来存放结点的前驱和后继结点信息，将大大降低存储空间的利用率。考查 n 个结点的二叉树，其中有 n+1 个空指针域，它们可以被用来存放“线索”，增加了线索的二叉树称为线索树（穿线树）。

设有一棵采用标准形式存储的二叉树 BT，对于 BT 中的每个结点 k，如它没有左（或右）子结点，而 k_1 是 k 的按中序遍历的前面（或后面）结点，则置结点 k 的左（或右）指针为 k_1 结点的指针。为了与 k 结点的真正子结点指针区别，另需在结点上增加两个标志域 ltag 和 rtag。如此改造后的线索树的结点类型定义如下：

```
typedef stuct BTnode{ / *穿线树结点类型 * /
char data;
struct node *lchild，*rchild;
int 1tag，rtag;
}BTNODE;
```

当 ltag=0 时，表示 lchild 指针指向其左孩子结点；当 ltag=1 时，表示 lchild 指针指向其前驱结点。当 rtag=0 时，表示 rchild 指针指向其右孩子结点；当 rtag=1 时，表示 rchild 指针指向其后继结点。

1.2.6　最优二叉树

树的路径长度是从树根到树中每一结点的路径长度之和。在结点数目相同的二叉树中，

完全二叉树的路径长度最短。在一些应用中，赋予树中结点的一个有某种意义的实数，这些数字称为结点的权。结点到树根之间的路径长度与该结点上权的乘积，称为结点的带权路径长度。树中所有叶结点的带权路径长度之和，称为树的带权路径长度（树的代价），通常记为：

$$\mathrm{WPL}=\sum_{i=1}^{n} w_i l_i$$

其中 n 表示叶子结点的数目，w_i 和 l_i 分别表示叶结点 k_i 的权值和根到结点 k_i 之间的路径长度。

在权值为 w_1，w_2，…，w_n 的 n 个叶子所构成的所有二叉树中，带权路径长度最小（即代价最小）的二叉树称为最优二叉树或哈夫曼树。

假设有 n 个权值，则构造出的哈夫曼树有 n 个叶子结点。n 个权值分别设为 w_1，w_2，⋯，w_n，则哈夫曼树的构造规则为：

①将 w_1，w_2，⋯，w_n 看成是有 n 棵树的森林（每棵树仅有一个结点）。

②在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点的权值之和。

③从森林中删除选取两棵树，并将新树加入森林。

④重复第②和③步，直到森林中只剩一棵树为止，该树即为所求的哈夫曼树。

例如，叶子结点的权值分别为 1、2、3、4、5、6，则构造哈夫曼树的过程如图 1-10 所示。

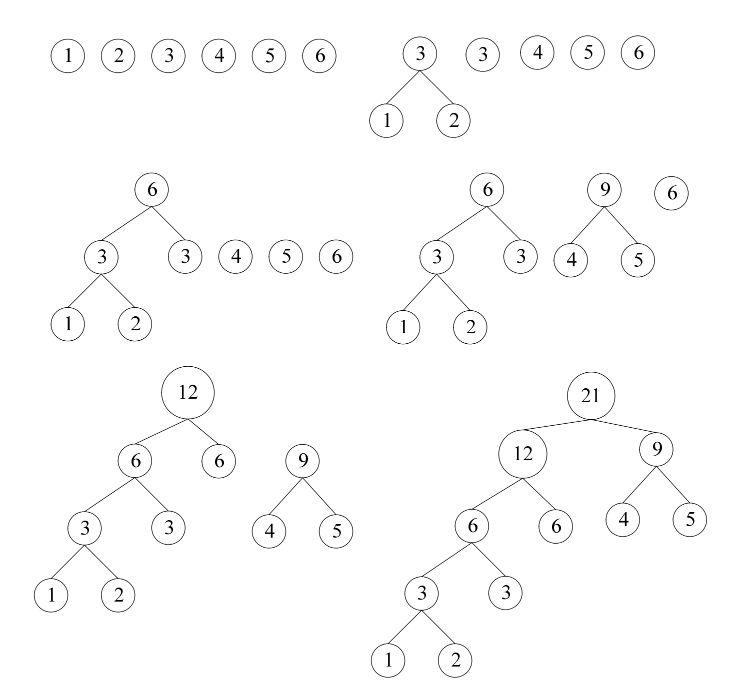


图 1-10　哈夫曼树的构造过程

在构造哈夫曼树的过程中，每次都是选取两棵最小权值的二叉树进行合并，因此使用的是贪心算法。

给定结点序列<c_i，p_i>（c_i 为编码字符，p_i 为 c_i 的频度），哈夫曼编码的过程如下：

- 用字符c_i作为叶子，p_i作为c_i的权，构造一棵哈夫曼树，并将树中左分支和右分支分别标记为0和1。
- 将从根到叶子的路径上的标号依次相连，作为该叶子所表示字符的编码。该编码即为最优前缀码。

给定字符集的哈夫曼树生成后，求哈夫曼编码的具体实现过程是依次以叶子结点 C[i]（$0 \leqslant i \leqslant n-1$）为出发点，向上回溯至根为止。上溯时走左分支则生成代码 0，走右分支则生成代码 1。需要注意如下几个问题。

- 由于生成的编码与要求的编码反序，将生成的代码先从后往前依次存放在一个临时串中，并设一个指针start指示编码在该串中的起始位置（start初始时指示串的结束位置）。
- 当某字符编码完成时，从临时串的start处将编码复制到该字符相应的位串bits中即可。
- 因为字符集大小为n，故变长编码的长度不会超过n，加上一个结束符“\0”，bits的大小应为n+1。

给定一个序列的集合，若不存在一个序列是另一个序列的前缀，则该序列集合称为前缀码。相反，给定一个序列的集合，若不存在一个序列是另一个序列的后缀，则该序列集合称为后缀码。平均码长或文件总长最小的前缀编码称为最优的前缀码，最优的前缀码对文件的压缩效果亦最佳。

$$\text{平均码长} = \sum_{i=1}^{n} p_i \, l_i$$

其中 p_i 为第 i 个字符的概率，l_i 为码长。

利用哈夫曼树很容易求出给定字符集及其概率（或频度）分布的最优前缀码。哈夫曼编码是一种应用广泛且非常有效的数据压缩技术。该技术一般可将数据文件压缩掉 20%至 90%，其压缩效率取决于被压缩文件的特征。

1.3 图

在线性结构（例如队列和栈）中，除第一个结点没有前驱，最后一个结点没有后继之外，每一个结点都有唯一的一个前驱和后继。在树形结构（例如树和二叉树）中，除根结点没有前驱外，一个结点只有一个前驱结点，但可以有若干个后继。在图结构中，一个结点的前驱和后继的个数都是任意的。

1.3.1 图的基础知识

1．图的基本概念

图 G 由两个集合 V 和 E 组成，记为 G=(V，E)。其中 V 是顶点的有穷非空集合，E 是 V 中顶点偶对（称为边）的有穷集合。通常，也将图 G 的顶点集和边集分别记为 $V(G)$和 $E(G)$。$E(G)$可以是空集。若 $E(G)$为空，则图 G 只有顶点而没有边。

图分为有向图和无向图两种。图 1-11（a）所示为一个有向图，在有向图中，一条有向边是由两个顶点组成的有序对，有序对通常用尖括号表示。<V_i，V_j>表示一条有向边，V_i 是边的始点（起点），V_j 是边的终点，<V_i，V_j>和<V_j，V_i>是两条不同的有向边。例如，在图 1-11（a）中，<V_1，V_4>和<V_4，V_1>是两条不同的边。有向边也称为弧，边的始点称为弧尾，终点称为弧头。

图 1-11（b）所示为一个无向图，无向图中的边均是顶点的无序对，无序对通常用圆括号表示。在无向图 G 中，如果 $i \neq j$，i、$j \in V$，$(i, j) \in E$，即 i 和 j 是 G 的两个不同的顶点，（i，j）是 G 中一条边，顶点 i 和 j 是相邻顶点，边（i，j）是与顶点 i 和 j 相关联的边。

如果限定任何一条边或弧的两个顶点都不相同，则有 n 个顶点的无向图至多有 $n(n-1)/2$ 条边，这样的无向图称为无向完全图。一个有向图至多有 $n(n-1)$条弧，这样的有向图称为有向完全图。

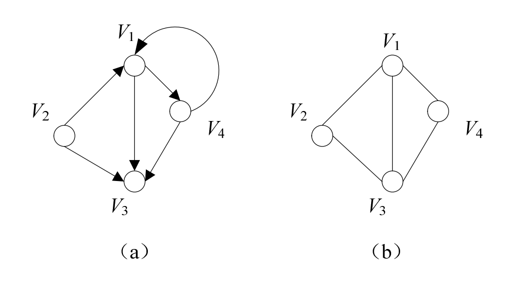


图 1-11　图的分类

如果同为无向图或同为有向图的两个图 G_1=（V_1，E_1）和 G_2=(V_2，E_2)满足 $V_2 \subseteq V_1$ 且 $E_2 \subseteq E_1$，则称图 G_2 是图 G_1 的子图。

在无向图中，一个顶点的度等于与其相邻接的顶点个数。在有向图中，一个顶点的入度等于邻接到该顶点的顶点个数，其出度等于邻接于该顶点的个数。

在图 G=（V，E）中，如果存在顶点序列（V_0，V_1，…，V_k）其中 V_0=P，V_k=Q，且（V_0，V_1），（V_1，V_2），…，（V_{k-1}，V_k）都在 E 中，则称顶点 P 到顶点 Q 有一条路径，并用（V_0，V_1，…，V_k）表示这条路径，路径的长度是路径的边数，这条路径的长度为 k。若 G 是有向图，则路径也是有向的。

在有向图 G 中，若对于 $V(G)$中任意两个不同的顶点 V_i 和 V_j，都存在从 V_i 到 V_j 及从 V_j 到 V_i 的路径，则称 G 是强连通图。

有向图的极大强连通子图称为 G 的强连通分量。强连通图只有一个强连通分量，即是其自身。非强连通的有向图有多个强连分量。

2．图的存储结构

最常用的图的存储结构有邻接矩阵和邻接表。

1）邻接矩阵

邻接矩阵反映顶点间的邻接关系，设 G=（V，E）是具有 n（$n \geqslant 1$）个顶点的图，G 的邻接矩阵 M 是一个 n 行 n 列的矩阵，并有若(i,j)或<i,j>$\in E$，则 $M[i][j]=1$；否则，$M[i][j]=0$。例如，图 1-11（a）和图 1-11（b）的邻接矩阵分别如下：

$$\boldsymbol{M}_{\mathrm{a}}=\begin{pmatrix}0&0&1&1\\1&0&1&0\\0&0&0&0\\1&0&1&0\end{pmatrix}\qquad \boldsymbol{M}_{\mathrm{b}}=\begin{pmatrix}0&1&1&1\\1&0&1&0\\1&1&0&1\\1&0&1&0\end{pmatrix}$$

由邻接矩阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵不一定对称。对于无向图，其邻接矩阵第 *i* 行元素的和即为顶点 *i* 的度。对于有向图，其邻接矩阵的第 *i* 行元素之和为顶点 *i* 的出度，而邻接矩阵的第 *j* 列元素之和为顶点 *j* 的入度。

若将图的每条边都赋上一个权，则称这种带权图为网（络）。如果图 *G*=（*V*，*E*）是一个网，若（*i*，*j*）或<*i*，*j*>∈*E*，则邻接矩阵中的元素 *M*[*i*][*j*]为（*i*，*j*）或<*i*，*j*>上的权。若（*i*，*j*）或<*i*，*j*>∉*E*，则 *M*[*i*][*j*]为无穷大或为大于图中任何权值的实数。

2）*邻接表*

在图的邻接表中，为图的每个顶点建立一个链表，且第 *i* 个链表中的结点代表与顶点 *i* 相关联的一条边或由顶点 *i* 出发的一条弧。有 *n* 个顶点的图，需用 *n* 个链表表示，这 *n* 个链表的头指针通常由顺序线性表存储。例如，图 1-11（a）和图 1-11（b）的邻接表分别如图 1-12（a）和图 1-12（b）所示。

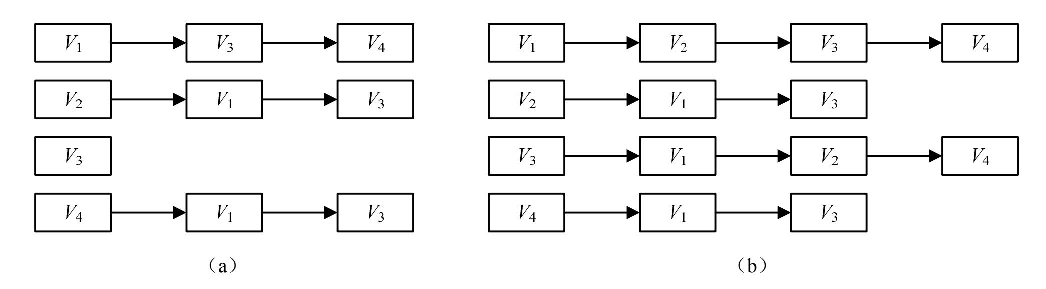


图 1-12　图的邻接表表示

在无向图的邻接表中，对应某结点的链表的结点个数就是该顶点的度。在有向图的邻接表中，对应某结点的链表的结点个数就是该顶点的出度。

3．图的遍历

和树的遍历类似，图的遍历也是从某个顶点出发，沿着某条搜索路径对图中每个顶点各做一次且仅做一次访问。它是许多图的算法的基础。深度优先遍历和广度优先遍历是最为重要的两种遍历图的方法。它们对无向图和有向图均适用。

1）*深度优先遍历*

在 *G* 中任选一顶点 *V* 为初始出发点（源点），则深度优先遍历可定义为首先访问出发点 *V*，并将其标记为已访问过；然后依次从 *V* 出发搜索 *V* 的每个邻接点 *W*。若 *W* 未曾访问过，则以 *W* 为新的出发点继续进行深度优先遍历，直至图中所有和源点 *V* 有路径相通的顶点（亦称为从源点可达的顶点）均已被访问为止。若此时图中仍有未访问的顶点，则另选一个尚未访问的顶点作为新的源点重复上述过程，直至图中所有顶点均已被访问为止。

图的深度优先遍历类似于树的前序遍历。对于无向图，如果图是连通的，那么按深度优先遍历时，可遍历全部顶点，得到全部顶点的一个遍历序列。如果遍历序列没有包含所有顶点，那么该图是不连通的。

2）*广度优先遍历*

广度优先的遍历过程是：首先访问出发顶点 V，然后访问与顶点 V 邻接的全部未被访问过的顶点 W_0，W_1，…，W_{k-1}；接着再依次访问与顶点 W_0，W_1，…，W_{k-1} 邻接的全部未被访问过的顶点。依次类推，直至图的所有顶点都被访问到，或出发顶点 V 所在的连通分量的全部顶点都被访问到为止。

从广度优先搜索遍历过程可知，若顶点 V 在顶点 W 之前被访问，则对 V 相邻的顶点的访问就先于只与 W 相邻的那些顶点的访问。因此，需要一个队列来存放被访问过的顶点，以便按顶点的访问顺序依次访问这些顶点相邻接的其他还未被访问过的顶点。

1.3.2 最小生成树

如果连通图 G 的一个子图是一棵包含 G 的所有顶点的树，则该子图称为 G 的生成树。生成树是连通图的包含图中的所有顶点的极小连通子图。值得注意的是，图的生成树并不唯一。从不同的顶点出发进行遍历，可以得到不同的生成树。

含有 n 个顶点的连通图的生成树有 n 个顶点和 n–1 条边。对一个带权的图（网），在一棵生成树中，各条边的权值之和称为这棵生成树的代价。其中代价最小的生成树称为最小代价生成树（简称最小生成树）。

MST 性质：设 $G=(V,E)$是一个连通网络，U 是顶点集 V 的一个真子集。若(u,v)是 G 中所有的一个端点在 U（$u\in U$）里、另一个端点不在 U（即 $v\in V-U$）里的边中，具有最小权值的一条边，则一定存在 G 的一棵最小生成树包括此边（u，v）。

求连通的带权无向图的最小代价生成树的算法有普里姆（Prim）算法和克鲁斯卡尔（Kruskal）算法。

1．普里姆算法

设已知 $G=(V,E)$ 是一个带权连通无向图，顶点 $V=\{0,1,2,\cdots,n-1\}$。设 U 是构造生成树过程中已被考虑在生成树上的顶点的集合。初始时，U 只包含一个出发顶点。设 T 是构造生成树过程中已被考虑在生成树上的边的集合，初始时 T 为空。如果边（i，j）具有最小代价，且 $i\in U$，$j\in$（$V-U$），那么最小代价生成树应包含边（i，j）。把 j 加到 U 中，把（i，j）加到 T 中。重复上述过程，直到 U 等于 V 为止。这时，T 即为要求的最小代价生成树的边的集合。

普里姆算法的特点是当前形成的集合 T 始终是一棵树。因为每次添加的边使树中的权尽可能小，因此这是一种贪心的策略。普里姆算法的时间复杂度为 $O(n^2)$，与图中边数无关，所以适合于稠密图。

2．克鲁斯卡尔算法

设 T 的初始状态只有 n 个顶点而无边的森林 $T=(V,\not\subset)$，按边长递增的顺序选择 E 中的 $n-1$ 安全边(u,v)并加入 T，生成最小生成树。所谓安全边，是指两个端点分别是森林 T 里两棵树中的顶点的边。加入安全边，可将森林中的两棵树连接成一棵更大的树，因为每一次添加到 T 中的边均是当前权值最小的安全边，MST 性质也能保证最终的 T 是一棵最小生成树。

克鲁斯卡尔算法的特点是当前形成的集合 T 除最后的结果外，始终是一个森林。克鲁斯卡尔算法的时间复杂度为 $O(e\log_2 e)$，与图中顶点数无关，所以较适合于稀疏图。

1.3.3 最短路径

带权图的最短路径问题即求两个顶点间长度最短的路径。其中路径长度不是指路径上边数的总和，而是指路径上各边的权值总和。路径长度的具体含义取决于边上权值所代表的意义。

1．单源最短路径

已知有向带权图（简称有向网）$G=(V, E)$，找出从某个源点 $s \in V$ 到 V 中其余各顶点的最短路径，称为单源最短路径。

目前，求单源最短路径主要使用迪杰斯特拉（Dijkstra）提出的一种按路径长度递增顺序产生各顶点最短路径的算法。若按长度递增的次序生成从源点 s 到其他顶点的最短路径，则当前正在生成的最短路径上除终点以外，其余顶点的最短路径均已生成（将源点的最短路径看作已生成的源点到其自身的长度为 0 的路径）。

迪杰斯特拉算法的基本思想是：设 S 为最短距离已确定的顶点集（看作红点集），$V-S$ 是最短距离尚未确定的顶点集（看作蓝点集）。

初始化：初始化时，只有源点 s 的最短距离是已知的（$SD(s)=0$），故红点集 $S=\{s\}$，蓝点集为空。

重复以下工作，按路径长度递增次序产生各顶点最短路径：在当前蓝点集中选择一个最短距离最小的蓝点来扩充红点集，以保证算法按路径长度递增的次序产生各顶点的最短路径。当蓝点集中仅剩下最短距离为∞的蓝点，或者所有蓝点已扩充到红点集时，s 到所有顶点的最短路径就求出来了。

需要注意的是：

- 若从源点到蓝点的路径不存在，则可假设该蓝点的最短路径是一条长度为无穷大的虚拟路径。
- 从源点s到终点v的最短路径简称为v的最短路径；s到v的最短路径长度简称为v的最短距离，并记为$SD(v)$。

根据按长度递增顺序产生最短路径的思想，当前最短距离最小的蓝点 k 的最短路径是：

源点红点 1，红点 2，…，红点 n，蓝点 k

距离为：源点到红点 n 最短距离 +<红点 n，蓝点 k>的边长。

为求解方便，可设置一个向量 $D[0,n-1]$，对于每个蓝点 $v \in V-S$，用 $D[v]$记录从源点 s 到达 v 且除 v 外中间不经过任何蓝点（若有中间点，则必为红点）的“最短”路径长度（简称估计距离）。若 k 是蓝点集中估计距离最小的顶点，则 k 的估计距离就是最短距离，即若 $D[k]=\min\{D[i]\ i \in V-S\}$，则 $D[k]=SD(k)$。

初始时，每个蓝点 v 的 $D[c]$值应为权 w<s，v>，且从 s 到 v 的路径上没有中间点，因为该路径仅含一条边<s，v>。

将 k 扩充到红点后，剩余蓝点集的估计距离可能由于增加了新红点 k 而减小，此时必须调整相应蓝点的估计距离。对于任意的蓝点 j，若 k 由蓝变红后使 $D[j]$变小，则必定是由于存在一条从 s 到 j 且包含新红点 k 的更短路径 P=<s，…，k,j>。且 $D[j]$减小的新路径 P 只可能是由路径<s，…，k>和边<k，j>组成的。所以，当 length(P)=$D[k]$+w<k，j>小于 $D[j]$时，应该用 P 的长度来修改 $D[j]$的值。

2．每一对顶点之间的最短路径

对图中每对顶点 u 和 v，找出 u 到 v 的最短路径问题。这一问题可用每个顶点作为源点调用一次单源最短路径问题的迪杰斯特拉算法予以解决。

但更常用的是弗洛伊德（Folyd）提出的求每一对顶点之间的最短路径的算法。设 G=（V，E）是有 n 个顶点的有向图，顶点集合 V={0，1，…，$n-1$}。图用邻接矩阵 M 表示。Floyd 算法的基本思想是递推地产生一个矩阵序列 C_0，C_1，C_2，…，C_n，其中 C_0 是已知的带权邻接矩阵 a，$C_k(i, j)$（$0 \leqslant i, j < n$）表示从顶点 i 到顶点 j 的中间顶点序号不大于 k 的最短路径长度。如果 i 到 j 的路径没有中间顶点，则对于 $0 \leqslant k < n$，有 $C_k(i,j) = C_0(i,j) = a[i][j]$。递推地产生 C_1，C_2，…，C_n 的过程就是逐步将可能是最短路径上的顶点作为路径上的中间顶点进行试探，直到为全部路径都找遍了所有可能成为最短路径上的中间顶点，所有的最短路径也就全部求出，算法就此结束。

设在第 k 次递推之前已求得 C_{k-1}（i，j）（$0 \leqslant i, j < n$），为求 $C_k(i, j)$需考虑如下两种情况。

- 如果从顶点 i 到顶点 j 的最短路径不经过顶点 k，则由 $C_k(i, j)$ 定义可知，从 i 到 j 中间的顶点序号不大于 k 的最短路径长度还是 $C_{k-1}(i, j)$，即 $C_k(i, j)=C_{k-1}(i, j)$。
- 如果从顶点 i 到顶点 j 的最短路径要经过顶点 k，则这样的一条路径是由 i 到 k 和由 k 到 j 的两条路径所组成的。由于 $C_{k-1}(i, k)$ 和 $C_{k-1}(k, j)$ 分别表示从 i 到 k 和从 k 到 j 的中间顶点序号不大于 k–1 的最短路径长度，若 $C_{k-1}(i, k)+C_{k-1}(k, j) < C_{k-1}(i, j)$，$C_{k-1}(i, k)+C_{k-1}(k, j)$ 必定是 i 到 j 的中间顶点序号不大于 k 的最短路径的长度，则 $C_k(i, j)= C_{k-1}(i, k)+C_{k-1}(k, j)$。

1.3.4　拓扑排序

对一个有向无环图 G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若 $\langle u, v\rangle \in E(G)$，则 u 在线性序列中出现在 v 之前。这样的线性序列称为满足拓扑次序的序列，简称拓扑序列。

要注意的是：

- 若将图中顶点按拓扑次序排成一行，则图中所有的有向边均是从左指向右的。
- 若图中存在有向环，则不可能使顶点满足拓扑次序。
- 一个有向无环图可能有多个拓扑序列。
- 当有向图中存在有向环时，拓扑序列不存在。

一个大工程有许多项目组，有些项目的实行则存在先后关系，某些项目必须在其他一些项目完成之后才能开始实行。工程项目实行的先后关系可以用一个有向图来表示，工程的项目称为活动，有向图的顶点表示活动，有向边表示活动之间开始的先后关系。这种有向图称为用顶点表示活动的网络，简称 AOV 网络，如图 1-13 所示。

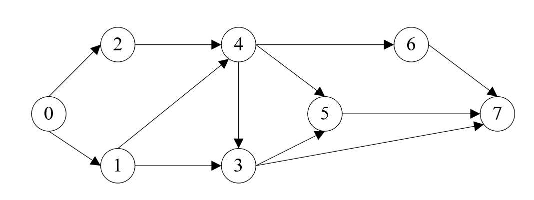


图 1-13　AOV 网络的例子

对 AOV 网络的顶点进行拓扑排序，就是对全部活动排成一个拓扑序列，使得在 AOV 网络中存在一条弧（i，j），则活动 i 排在活动 j 之前。例如，对图 1-13 中的有向图的顶点

进行拓扑排序，可以得到多个不同的拓扑序列，如 02143567，02143657，01243567 等。

1.3.5 关键路径

在 AOV 网络中，如果边上的权表示完成该活动所需的时间，则称这样的 AOV 为 AOE 网络。例如，图 1-14 所示为一个具有 10 个活动的某个工程的 AOE 网络。图中有 7 个顶点，分别表示事件 1～7，其中 1 表示工程开始状态，7 表示工程结束状态，边上的权表示完成该活动所需的时间。

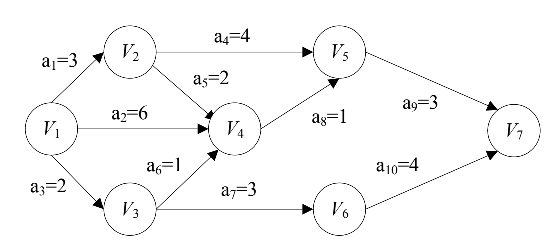


图 1-14　AOE 网络的例子

因 AOE 网络中的某些活动可以并行地进行，所以完成工程的最少时间是从开始顶点到结束顶点的最长路径长度，称从开始顶点到结束顶点的最长路径为关键路径（临界路径），关键路径上的活动为关键活动。

为了找出给定的 AOE 网络的关键活动，从而找出关键路径，先定义几个重要的量。

$V_e(j)$、$V_l(j)$ ：顶点 j 事件最早、最迟发生的时间。

$e(i)$、$l(i)$： 活动 i 最早、最迟开始的时间。

从源点 V_1 到某顶点 V_j 的最长路径长度，称为事件 V_j 的最早发生时间，记为 $V_e(j)$。$V_e(j)$ 也是以 V_j 为起点的出边 $<V_j,V_k>$ 所表示的活动 a_i 的最早开始时间 $e(i)$。

在不推迟整个工程完成的前提下，一个事件 V_j 允许的最迟发生时间，记为 $V_l(j)$。显然，$l(i)=V_l(j)-$（a_i 所需时间），其中 j 为 a_i 活动的终点。满足条件 $l(i)=e(i)$ 的活动为关键活动。

求顶点 V_j 的 $V_e(j)$ 和 $V_l(j)$ 可按如下两步来做。

①由源点开始向汇点递推

$$\begin{cases} V_e(1)=0 \\ V_e(j)=\text{MAX}\{V_e(i)+\text{d}(i,j)\}, <V_i,\ V_j>\in E_1, 2\leqslant j\leqslant n \end{cases}$$

其中，E_1 是网络中以 V_j 为终点的入边集合。

②由汇点开始向源点递推

$$\begin{cases} V_l(n)=V_e(n) \\ V_l(j)=\text{MIN}\{V_l(k)-d(j,k)\}, <V_j,\ V_k>\in E_2, 2\leqslant j\leqslant n-1 \end{cases}$$

其中，E_2 是网络中以 V_j 为起点的出边集合。

要求一个 AOE 的关键路径，一般需要根据上述变量列出一张表格，逐个检查。例如，求如图 1-14 所示的 AOE 关键路径的过程如表 1-1 所示。

表 1-1　求关键路径的过程

顶点	$V_e(j)$	$V_l(j)$	边	$e(i)$	$l(i)$	$l(i)-e(i)$
V_1	0	0	$a_1(3)$	0	0	0
V_2	3	3	$a_2(6)$	0	0	0
V_3	2	3	$a_3(2)$	0	1	1
V_4	6	6	$a_4(4)$	3	3	0
V_5	7	7	$a_5(2)$	3	4	1
V_6	5	6	$a_6(1)$	2	5	3
V_7	10	10	$a_7(3)$	2	3	1
			$a_8(1)$	6	6	0
			$a_9(3)$	7	7	0
			$a_{10}(4)$	5	6	1

因此，图 1-14 的关键活动为 a_1，a_2，a_4，a_8 和 a_9（即表 1-1 阴影所示活动），其对应的关键路径有两条，分别为（V_1，V_2，V_5，V_7）和（V_1，V_4，V_5，V_7），长度都是 10。

1.4　排序

所谓排序，就是要整理文件中的记录，使之按关键字递增（或递减）次序排列起来。当待排序记录的关键字均不相同时，排序结果是唯一的，否则排序结果不唯一。

在待排序的文件中，若存在多个关键字相同的记录，经过排序后这些具有相同关键字的记录之间的相对次序保持不变，该排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生变化，则称这种排序方法是不稳定的。

要注意的是，排序算法的稳定性是针对所有输入实例而言的。即在所有可能的输入实例中，只要有一个实例使得算法不满足稳定性要求，则该排序算法就是不稳定的。

1.4.1　插入排序

插入排序的基本思想是每步将一个待排序的记录按其排序码值的大小，插到前面已经排好的文件中的适当位置，直到全部插入完为止。插入排序方法主要有直接插入排序和希尔排序。

1．直接插入排序

直接插入排序的过程为在插入第 i 个记录时，R_1，R_2，…，R_{i-1} 已经排好序，将第 i 个记录的排序码 k_i 依次和 R_1，R_2，…，R_{i-1} 的排序码逐个进行比较，找到适当的位置。使用直接插入排序，对于具有 n 个记录的文件，要进行 n–1 趟排序。各种状态下的时间复杂度如表 1-2 所示。

表 1-2　直接插入排序有关数据

初始文件状态	正　　序	反　　序	无序（平均）
第 i 趟的关键字比较次数	1	$i+1$	$(i-2)/2$
总关键字比较次数	$n-1$	$(n+2)(n-1)/2$	$n^2/4$

续表

初始文件状态	正　　序	反　　序	无序（平均）
第 i 趟记录移动次数	0	i+2	(i–2)/2
总的记录移动次数	0	$(n-1)(n+4)/2$	$n^2/4$
时间复杂度	$O(n)$	$O(n^2)$	$O(n^2)$

说明：初始文件按关键字递增有序，简称“正序”，初始文件按关键字递减有序，简称“反序”。

2．希尔排序

希尔（Shell）排序的基本思想是：先取一个小于 n 的整数 d_1 作为第一个增量，把文件的全部记录分成 d_1 个组。所有距离为 d_1 的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量 $d_2<d_1$ 重复上述的分组和排序，直至所取的增量 $d_t=1(d_t<d_{t-1}<O<d_2<d_1)$，即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。

一般取 $d_1=n/2$，$d_{i+1}=d_i/2$。如果结果为偶数，则加 1，保证 d_i 为奇数。

例如，要对关键码{72，28，51，17，96，62，87，33，45，24}进行排序，这里 n=10，首先取 $d_1=n/2=5$，则排序过程如图 1-15 所示。

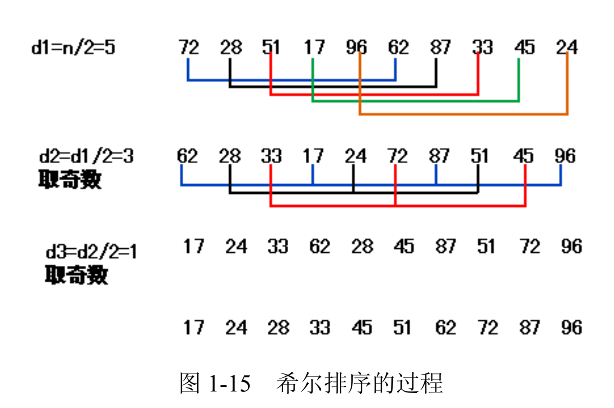


图 1-15　希尔排序的过程

希尔排序是不稳定的，希尔排序的执行时间依赖于增量序列，其平均时间复杂度为 $O(n^{1.3})$。

1.4.2　选择排序

选择排序的基本思想是每步从待排序的记录中选出排序码最小的记录，顺序存放在已排序的记录序列的后面，直到全部排完。选择排序中主要使用直接选择排序和堆排序。

1．直接选择排序

直接选择排序的过程是，首先在所有记录中选出排序码最小的记录，把它与第 1 个记录交换，然后在其余的记录内选出排序码最小的记录，与第 2 个记录交换……依次类推，直到所有记录排完为止。

无论文件初始状态如何，在第 i 趟排序中选出最小关键字的记录，需做 $n-i$ 次比较，因此，总的比较次数为 $n(n-1)/2=O(n^2)$。当初始文件为正序时，移动次数为 0；文件初态为反序时，每趟排序均要执行交换操作，总的移动次数取最大值 $3(n-1)$。直接选择排序的平均时间复杂度为 $O(n^2)$。直接选择排序是不稳定的。

2．堆排序

堆排序是一种树形选择排序，是对直接选择排序的有效改进。n 个关键字序列 $K_1, K_2, \cdots, K_n$ 称为堆，当且仅当该序列满足（$K_i \leqslant K_{2i}$ 且 $K_i \leqslant K_{2i+1}$）或（$K_i \geqslant K_{2i}$ 且 $K_i \geqslant K_{2i+1}$），（$1 \leqslant i \leqslant \lfloor n/2 \rfloor$）。根结点（堆顶）的关键字是堆里所有结点关键字中最小者，称为小根堆；根结点的关键字是堆里所有结点关键字中最大者，称为大根堆。

若将此序列所存储的向量 $R[1 \cdots n]$看作一棵完全二叉树的存储结构，则堆实质上是满足如下性质的完全二叉树，即树中任一非叶结点的关键字均不大于（或不小于）其左、右孩子（若存在）结点的关键字。

堆排序的关键步骤有两个，一是如何建立初始堆；二是当堆的根结点与堆的最后一个结点交换后，如何对少了一个结点后的结点序列做调整，使之重新成为堆。

下面通过一个例子来具体说明建立初始堆和调整堆的过程。假设关键字序列为{42，13，24，91，23，16，05，88}，则第一次建立的二叉树如图 1-16（a）所示。

①从 $i = [n/2]$开始比较父结点和子结点的关系，如果不满足堆的定义，就进行调整。假设需要建立大根堆，本题 n=8，所以从第 4 个元素（91）开始调整。

- 因为91大于其子结点88，所以不需要调整。
- 再看第3个元素（24），同样，因为24大于其子结点16和05，也不需要调整。
- 再看第2个元素（13），13小于其子结点23和91，需要把13和91交换（把父结点与关键值最大的那个子结点交换）。这时，13比其子结点88要小，又需要交换。结果如图1-16（b）所示。

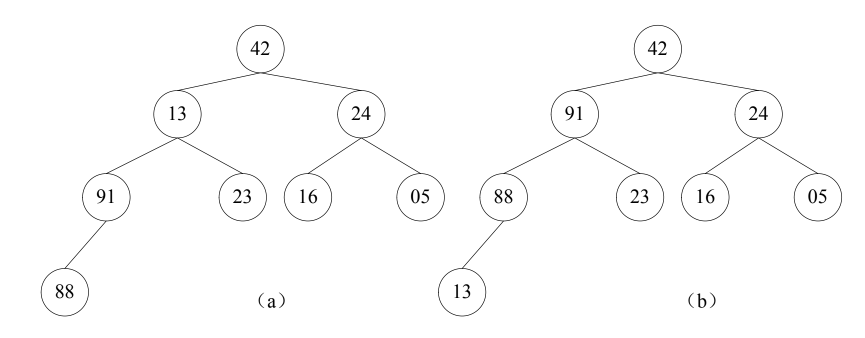


图 1-16　建立堆的过程

- 再看第1个元素（42），因为42小于其左子结点91，需要交换。
- 这时，42还小于其左子结点88，又需要交换42和88的值。建堆过程结束，所建立的初始堆如图1-17（a）所示。

②在初始堆的基础上，把第一个元素（91）和最后一个元素（13）交换，输出 91。这时，如图 1-17（b）所示。

③在图 1-17（b）的基础上，因 13 小于其左、右子结点 88 和 24，则和 88 交换，交换后，13 还小于其左、右子结点 42 和 23，则和 42 再交换，如图 1-18（a）所示。

④图 1-18（a）所示为一个 $n-1$ 个元素的堆，把第一个元素（88）和最后一个元素（05）交换，输出 88，如图 1-18（b）所示。

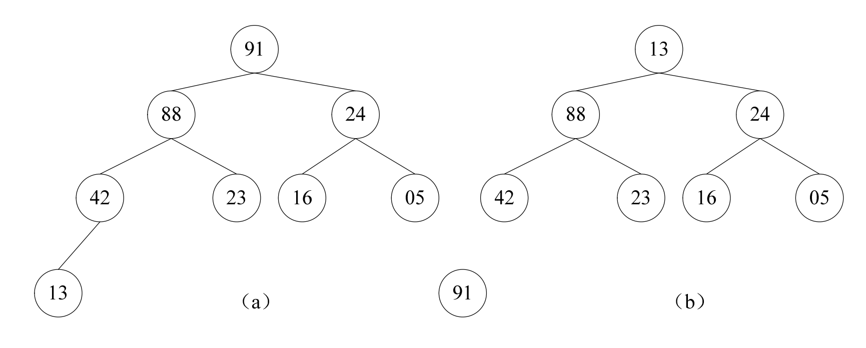


图 1-17　初始堆及调整一

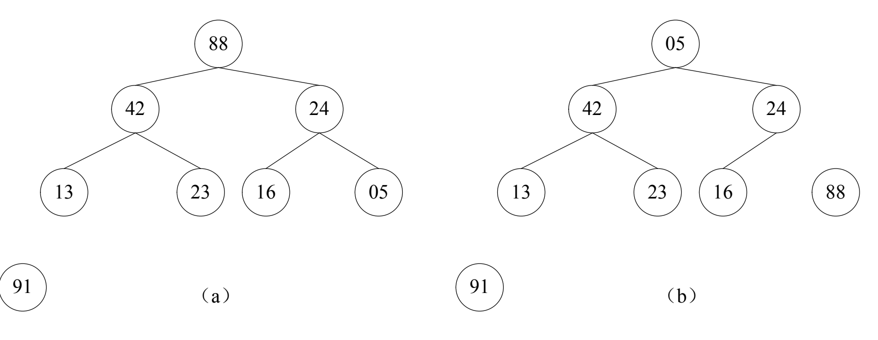


图 1-18　调整堆的过程之一

⑤在图 1-18（b）的基础上，因 05 小于其左、右子结点 42 和 24，则和 42 交换，交换后，05 还小于其左、右子结点 13 和 23，则和 23 再交换，如图 1-19（a）所示。

⑥图 1-19（a）所示为一个 $n-2$ 个元素的堆，把第一个元素（42）和最后一个元素（16）交换，输出 42，如图 1-19（b）所示。

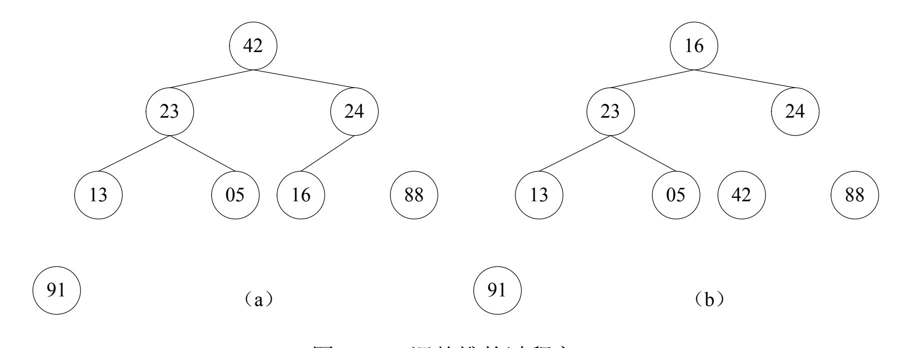


图 1-19　调整堆的过程之二

⑦在图 1-19（b）的基础上，因 16 小于其左、右子结点 23 和 24，则和 24 交换，如图 1-20（a）所示。

⑧图 1-20（a）所示为一个 $n-3$ 个元素的堆，把第一个元素（24）和最后一个元素（05）交换，输出 24，如图 1-20（b）所示。

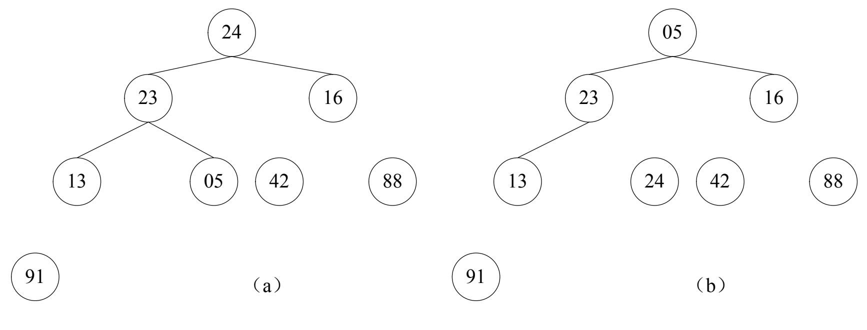


图 1-20　调整堆的过程之三

⑨在图 1-20（b）的基础上，因 05 小于其左、右子结点 23 和 16，则和 23 交换。交换后，05 还是小于其子结点 13，和 13 再交换，如图 1-21（a）所示。

⑩图 1-21（a）所示为一个 $n-4$ 个元素的堆，把第一个元素（23）和最后一个元素（05）交换，输出 23，如图 1-21（b）所示。

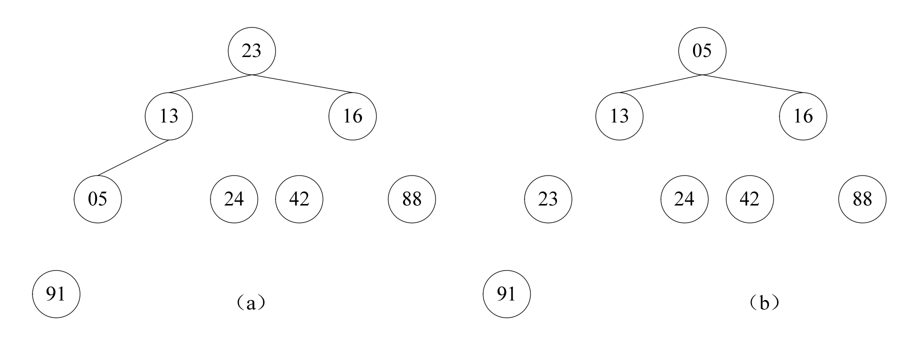


图 1-21　调整堆的过程之四

⑪在图 1-21（b）的基础上，因 05 小于其左、右子结点 13 和 16，则和 16 交换，如图 1-22（a）所示。

⑫图 1-22（a）所示为一个 $n-5$ 个元素的堆，把第一个元素（16）和最后一个元素（05）交换，输出 16，如图 1-22（b）所示。

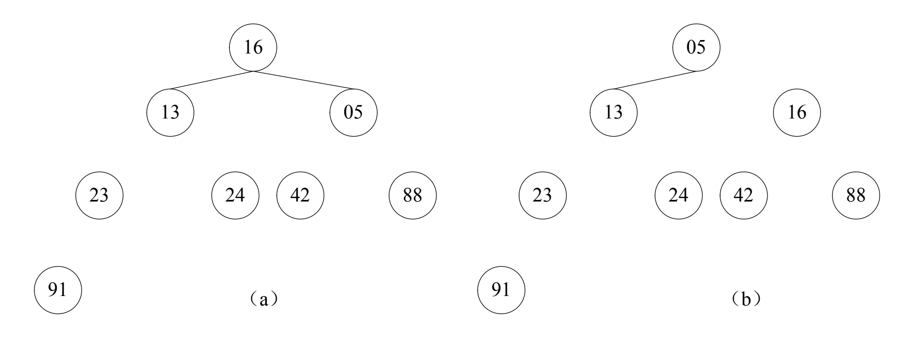


图 1-22　调整堆的过程之五

⑬在图 1-22（b）的基础上，因 05 小于其左子结点 13，则和 13 交换，如图 1-23（a）所示。

⑭图 1-23（a）所示为一个 $n-6$ 个元素的堆，把第一个元素（13）和最后一个元素（05）交换，输出 13，如图 1-23（b）所示，堆排序过程结束。

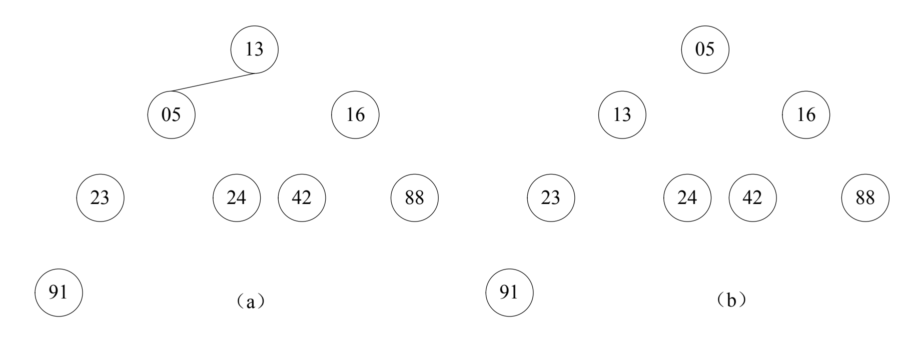


图 1-23　调整堆的过程之六

堆排序的最坏时间复杂度为 $O(n\log_2 n)$，堆排序的平均性能较接近于最坏性能。由于建初始堆所需的比较次数较多，所以堆排序不适宜于记录数较少的文件。堆排序是就地排序，辅助空间为 $O(1)$，它是不稳定的排序方法。

1.4.3　交换排序

交换排序的基本思想是两两比较待排序记录的排序码，并交换不满足顺序要求的那些偶对，直到满足条件为止。交换排序的主要方法有冒泡排序和快速排序。

1．冒泡排序

冒泡排序将被排序的记录数组 $R[1\cdots n]$垂直排列，每个记录 $R[i]$看作重量为 k_i 的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组 R：凡扫描到违反本原则的轻气泡，就使其向上“飘浮”。如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

冒泡排序的具体过程如下。

第一步，先比较 k_1 和 k_2，若 $k_1>k_2$，则交换 k_1 和 k_2 所在的记录，否则不交换。继续对 k_2 和 k_3 重复上述过程，直到处理完 k_{n-1} 和 k_n。这时最大的排序码记录转到了最后位置，称第 1 次起泡，共执行 $n-1$ 次比较。

与第一步类似，从 k_1 和 k_2 开始比较，到 k_{n-2} 和 k_{n-1} 为止，共执行 $n-2$ 次比较，称第 2 次起泡。

依次类推，共做 $n-1$ 次起泡，完成整个排序过程。

若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数为 $n-1$ 次，记录移动次数为 0。因此，冒泡排序最好的时间复杂度为 $O(n)$。

若初始文件是反序的，需要进行 $n-1$ 趟排序。每趟排序要进行 $n-i$ 次关键字的比较（$1\leqslant i\leqslant n-1$），且每次比较都必须移动记录三次来达到交换记录位置。在这种情况下，比较次数达到最大值 $n(n-1)/2=O(n^2)$，移动次数也达到最大值 $3n(n-1)/2=O(n^2)$。因此，冒泡排序的最坏时间复杂度为 $O(n^2)$。

虽然冒泡排序不一定要进行 $n-1$ 趟，但由于它的记录移动次数较多，故平均时间性能比直接插入排序要差得多。冒泡排序是就地排序，且它是稳定的。

2．快速排序

快速排序采用了一种分治的策略，通常称其为分治法。其基本思想是将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

快速排序的具体过程如下。

第一步，在待排序的 n 个记录中任取一个记录，以该记录的排序码为准，将所有记录分成两组，第 1 组各记录的排序码都小于等于该排序码，第 2 组各记录的排序码都大于该排序码，并把该记录排在这两组中间。

第二步，采用同样的方法，对左边的组和右边的组进行排序，直到所有记录都排到相应的位置为止。

例如，要对关键码{7，2，5，1，9，6，8，3}进行排序，选择第一个元素为基准。第一趟排序的过程如图 1-24 所示。

要注意的是，在快速排序中，选定了以第一个元素为基准，接着就拿最后一个元素和第一个元素比较，如果大于第一个元素，则保持不变，再拿倒数第二个元素和基准比较；如果小于基准，则进行交换。交换之后，再从前面的元素开始与基准比较，如果小于基准，则保持不变；如果大于基准，则交换。交换之后，再从后面开始比较，依次类推，前后交叉进行。

然后，再采取同样的办法对{3，2，5，1，6}和{8，9}分别进行排序，具体过程如图 1-25 所示。

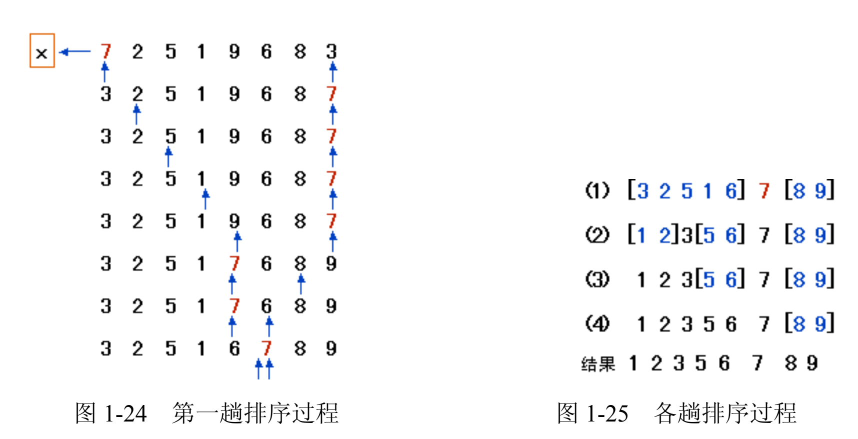


图 1-24　第一趟排序过程　　　图 1-25　各趟排序过程

快速排序的时间主要耗费在划分操作上，对长度为 k 的区间进行划分，共需 $k-1$ 次关键字的比较。

最坏情况是每次划分选取的基准都是当前无序区中关键字最小（或最大）的记录，划分的结果是基准左边的子区间为空（或右边的子区间为空），而划分所得的另一个非空的子区间中的记录数目，仅仅比划分前的无序区中的记录个数减少一个。因此，快速排序必须做 $n-1$ 次划分，第 i 次划分开始时区间长度为 $n-i+1$，所需的比较次数为 $n-i$（$1 \leqslant i \leqslant n-1$），故总的比较次数达到最大值 $n(n-1)/2=O(n^2)$。如果按上述给出的划分算法，以每次取当前无序区的第 1 个记录为基准，那么当文件的记录已按递增序（或递减序）排列时，每次划分所取的基准就是当前无序区中关键字最小（或最大）的记录，则快速排序所需的比较次数反而最多。

在最好情况下，每次划分所取的基准都是当前无序区的“中值”记录，划分的结果是基准的左、右两个无序子区间的长度大致相等。总的关键字比较次数 $O(n\log_2 n)$。

因为快速排序的记录移动次数不大于比较的次数，所以快速排序的最坏时间复杂度应为 $O(n^2)$，最好时间复杂度为 $O(n\log_2 n)$。

尽管快速排序的最坏时间为 $O(n^2)$，但就平均性能而言，它是基于关键字比较的内部排序算法中速度最快者，快速排序亦因此而得名。它的平均时间复杂度为 $O(n\log_2 n)$。快速排序在系统内部需要一个栈来实现递归。若每次划分较为均匀，则其递归树的高度为 $O(\log_2 n)$，故递归后需栈空间为 $O(\log_2 n)$。在最坏情况下，递归树的高度为 $O(n)$，所需的栈空间为 $O(n)$。快速排序是不稳定的。

1.4.4　归并排序

归并排序是将两个或两个以上的有序子表合并成一个新的有序表。初始时，把含有 n 个结点的待排序序列看作由 n 个长度都为 1 的有序子表所组成，将它们依次两两归并得到长度为 2 的若干有序子表，再对它们两两合并。直到得到长度为 n 的有序表，排序结束。

例如，需要对关键码{72，28，51，17，96，62，87，33}进行排序，其归并过程如图 1-26 所示。

72　28　51　17　96　62　87　33

72　28　51　17　96　62　87　33

[28 72]　[17 51]　[62 96]　[33 87]

[28 72]　[17 51]　[62 96]　[33 87]

[17　28　51　72]　[62　33　87　96]

[17　28　33　51　62　72　87　96]

图 1-26　归并排序的过程

归并排序是一种稳定的排序，可用顺序存储结构，也易于在链表上实现。对长度为 n 的文件，需进行 $\log_2 n$ 趟二路归并，每趟归并的时间为 $O(n)$，故其时间复杂度无论在最好情况下还是在最坏情况下均是 $O(n\log_2 n)$。归并排序需要一个辅助向量来暂存两个有序子文件归并的结果，故其辅助空间复杂度为 $O(n)$，显然它不是就地排序。

1.4.5　基数排序

设单关键字的每个分量的取值范围均是 $C_0 \leqslant k_j \leqslant C_{rd-1}(0 \leqslant j < rd)$，可能的取值个数 rd 称为基数。基数的选择和关键字的分解因关键字的类型而异。

- 若关键字是十进制整数，则按个、十等位进行分解，基数 rd=10，C_0=0，C_9=9，d为最长整数的位数。
- 若关键字是小写的英文字符串，则 rd=26，C_o='a'，C_{25}='z'，d为字符串的最大长度。

基数排序的基本思想是从低位到高位依次对待排序的关键码进行分配和收集，经过 d 趟分配和收集，就可以得到一个有序序列。

基数排序的具体实现过程如下。

设有 r 个队列，队列的编号分别为 0，1，2，…，$r-1$。首先按最低有效位的值把 n 个关键字分配到这 r 个队列中，然后从小到大将各队列中的关键字再依次收集起来；接着

再按次低有效位的值把刚刚收集起来的关键字分配到 r 个队列中。重复上述收集过程，直至最高有效位，这样便得到一个从小到大的有序序列。为减少记录移动的次数，队列可以采用链式存储分配，称为链式基数排序。每个队列设有两个指针，分别指向队头和队尾。

例如，需要对{288，371，260，531，287，235，56，299，18，23}进行排序，因为这些数据最高位为百位，所以需要分三趟分配与收集。第一趟分配与收集（按个位数）的过程如图 1-27 所示。第二趟分配与收集（按十位数）的过程如图 1-28 所示。第三趟分配与收集（按百位数）的过程如图 1-29 所示。

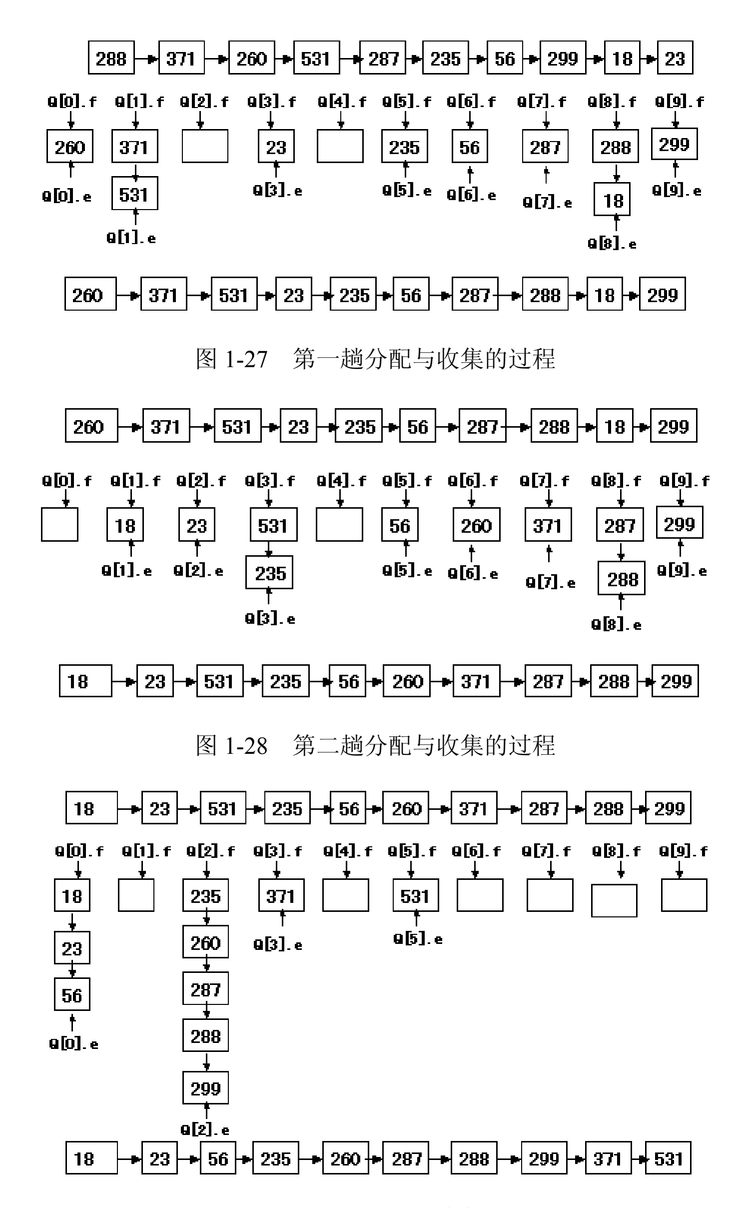


图 1-27　第一趟分配与收集的过程

图 1-28　第二趟分配与收集的过程

图 1-29　第三趟分配与收集的过程

基数排序的时间复杂度为 $O(d(r+n))$，所需的辅助存储空间为 $O(n+rd)$，基数排序是稳定的。

1.4.6 算法复杂性比较

在此，把常用的排序算法的复杂度进行列表，如表 1-3 所示。

表 1-3 排序算法时间复杂度表

排 序 方 法	最 好 情 况	平 均 时 间	最 坏 情 况	辅 助 空 间	稳定性
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	O(1)	√
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	√
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	O(1)	√
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	×
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	O(1)	×
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	√
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(d(n+rd))$	O(rd)	√

注：rd 称为基数，基数的选择和关键字的分解因关键字的类型而异。

1.5 查找

查找是指给定一个值 k，在含有 n 个结点的表中找出关键字等于给定值 k 的结点。若找到，则查找成功，返回该结点的信息或该结点在表中的位置；否则查找失败，返回相关的指示信息。若在查找的同时对表做修改操作（如插入和删除），则相应的表称为动态查找表，否则称为静态查找表。

查找运算的主要操作是关键字的比较，所以通常把查找过程中对关键字需要执行的平均比较次数（也称为平均查找长度）作为衡量一个查找算法效率优劣的标准。平均查找长度 ASL 定义为：

$$\mathrm{ASL}=\sum_{i=1}^{n} p_i c_i$$

其中 n 是结点的个数，p_i 是查找第 i 个结点的概率。若不特别声明，则认为每个结点的查找概率相等，即 $p_1=p_2=\cdots=p_n=1/n$；c_i 是找到第 i 个结点所需进行的比较次数。

1.5.1 顺序查找

顺序查找的基本思想是从表的一端开始，顺序扫描线性表，依次将扫描到的结点关键字和给定值 k 相比较。若当前扫描到的结点关键字与 k 相等，则查找成功；若扫描结束后，仍未找到关键字等于 k 的结点，则查找失败。顺序查找方法既适用于线性表的顺序存储结构，也适用于线性表的链式存储结构。

成功时的顺序查找的平均查找长度如下：

$$\mathrm{ASL}=\sum_{i=1}^{n} p_i c_i=\sum_{i=1}^{n}(n-i+1)=np_1+(n-1)p_2+\cdots+2p_{n-1}+p_n$$

在等概率情况下，$p_i=1/n(1\leqslant i\leqslant n)$，故成功的平均查找长度为$(n+\cdots+2+1)/n=(n+1)/2$，即查找成功时的平均比较次数约为表长的一半。若 k 值不在表中，则需进行（$n+1$）次比较之后才能确定查找失败。

若事先知道表中各结点的查找概率不相等和它们的分布情况，则应将表中结点按查找概率由小到大地存放，以便提高顺序查找的效率。

顺序查找的优点是算法简单，且对表的结构无任何要求，无论是用向量还是用链表来存放结点，也无论结点之间是否按关键字有序，它都同样适用。缺点是查找效率低，因此，当 n 较大时不宜采用顺序查找。

1.5.2 二分法查找

二分法查找又称为折半查找，它是一种效率较高的查找方法。二分法查找要求线性表是有序表，即表中结点按关键字有序，并且要用向量作为表的存储结构。

二分法查找的基本思想是：（设 R[low，…，high]是当前的查找区间）

①确定该区间的中点位置：mid=[(low+high)/2]。

②将待查的 k 值与 R[mid].key 比较，若相等，则查找成功并返回此位置，否则需确定新的查找区间，继续二分查找，具体方法如下。

- 若R[mid].key＞k，则由表的有序性可知R[mid，…，n].key均大于k，因此若表中存在关键字等于k的结点，则该结点必定是在位置mid左边的子表R[low，…，mid－1]中。因此，新的查找区间是左子表R[low，…，high]，其中high=mid－1。
- 若R[mid].key＜k，则要查找的k必在mid的右子表R[mid+1，…，high]中，即新的查找区间是右子表R[low，…，high]，其中low=mid+1。
- 若R[mid].key=k，则查找成功，算法结束。

③下一次查找是针对新的查找区间进行的，重复步骤①和②。

④在查找过程中，low 逐步增加，而 high 逐步减少。如果 high＜low，则查找失败，算法结束。

因此，从初始的查找区间 R[1，…，n]开始，每经过一次与当前查找区间的中点位置上的结点关键字的比较，就可确定查找是否成功，不成功则当前的查找区间就缩小一半。重复这一过程直至找到关键字为 k 的结点，或者直至当前的查找区间为空（即查找失败）时为止。

例如，要在{11，13，17，23，31，36，40，47，52，58，66，73，77，82，96，99}中查找 58 的过程如图 1-30 所示（粗体表示 mid 位置）。在上述序列中查找 35 的过程如图 1-31 所示。

11 13 17 23 31 36 40 **47** 52 58 66 73 77 82 96 99

11 13 17 23 31 36 40 47 52 58 66 **73** 77 82 96 99

11 13 17 23 31 36 40 47 52 **58** 66 73 77 82 96 99

图 1-30 二分法查找 58

11 13 17 23 31 36 40 **47** 52 58 66 73 77 82 96 99

11 13 17 **23** 31 36 40 47 52 58 66 73 77 82 96 99

11 13 17 23 31 **36** 40 47 52 58 66 73 77 82 96 99

11 13 17 23 **31** 36 40 47 52 58 66 73 77 82 96 99

图 1-31 二分法查找 35

二分法查找过程可用二叉树来描述。把当前查找区间的中间位置上的结点作为根，左子表和右子表中的结点分别作为根的左子树和右子树。由此得到的二叉树，称为描述二分查找的判定树或比较树。要注意的是，判定树的形态只与表结点个数 n 相关，而与输入实例中 R[1，…，n].key 的取值无关。

设内部结点的总数为 $n=2^h-1$，则判定树是深度为 $h=\log_2(n+1)$ 的满二叉树。树中第 k 层上的结点个数为 2^{k-1}，查找它们所需的比较次数是 k。因此在等概率假设下，二分法查找成功时的平均查找长度为 $\log_2(n+1)-1$。二分法查找在查找失败时所需比较的关键字个数不超过判定树的深度，在最坏情况下查找成功的比较次数也不超过判定树的深度，即为 $[\log_2(n+1)]$。二分法查找的最坏性能和平均性能相当接近。

虽然二分法查找的效率高，但是要将表按关键字排序。而排序本身是一种很费时的运算。即使采用高效率的排序方法也要花费 $O(n\log_2 n)$ 的时间，二分法查找只适用于顺序存储结构。为保持表的有序性，在顺序结构里插入和删除都必须移动大量的结点。因此，二分法查找特别适用于那种一经建立就很少改动而又经常需要查找的线性表。

对那些查找少而又经常需要改动的线性表，可采用链表作为存储结构，进行顺序查找。链表上无法实现二分法查找。

1.5.3　分块查找

分块查找（Blocking Search）又称为索引顺序查找。它是一种性能介于顺序查找和二分查找之间的查找方法。二分查找表由分块有序的线性表和索引表组成。表 $R[1,\cdots,n]$ 均分为 b 块，前 $b-1$ 块中结点个数为 $s=[n/b]$，第 b 块的结点数允许小于等于 s；每一块中的关键字不一定有序，但前一块中的最大关键字必须小于后一块中的最小关键字，即表是分块有序的。

抽取各块中的最大关键字及其起始位置构成一个索引表 $ID[1,\cdots,b]$，即 $ID[i](1\leqslant i\leqslant b)$ 中存放第 i 块的最大关键字及该块在表 R 中的起始位置。由于表 R 是分块有序的，所以索引表是一个递增有序表。

分块查找的基本思想是索引表是有序表，可采用二分查找或顺序查找，以确定待查的结点在哪一块。

由于块内无序，只能用顺序查找。分块查找是两次查找过程。整个查找过程的平均查找长度是两次查找的平均查找长度之和。如果以二分查找来确定块，则分块查找成功时的平均查找长度为 $\text{ASL1}=\log_2(b+1)-1+(s+1)/2\approx\log_2(n/s+1)+s/2$；如果以顺序查找确定块，分块查找成功时的平均查找长度为 $\text{ASL2}=(b+1)/2+(s+1)/2=(s^2+2s+\text{n})/(2s)$。

注意： 当 $s=\sqrt{n}$ 时，ASL2 取极小值 $\sqrt{n}+1$，即当采用顺序查找确定块时，应将各块中的结点数选定为 $\sqrt{n}$。

分块查找的优点是在表中插入或删除一个记录时，只要找到该记录所属的块，就在该块内进行插入和删除运算；因块内记录的存放是任意的，所以插入或删除比较容易，无须移动大量记录。

分块查找的主要代价是增加一个辅助数组的存储空间和将初始表分块排序的运算。

1.5.4　散列表

散列表又称为杂凑表，是一种非常实用的查找技术，能在 $O(1)$ 时间内完成查找。

将所有可能出现的关键字集合记为 U（简称全集）。实际发生（即实际存储）的关键字集合记为 K（$|K|$ 比 $|U|$ 小得多）。散列方法是使用函数 h 将 U 映射到表 $T[0,\cdots,m-1]$ 的下标上（$m=O(|U|)$）。这样以 U 中的关键字为自变量，以 h 为函数的运算结果就是相应结点的存储地址。从而达到在 $O(1)$ 时间内就可完成查找。

- h：U→{0，1，2，…，$m-1$}，通常称h为散列函数（Hash函数）。散列函数h的作用是压缩待处理的下标范围，使待处理的$|U|$个值减少到m个值，从而降低空间开销。
- T为散列表（Hash Table）。
- $h(K_i)$（$K_i \in U$）是关键字为K_i的结点存储地址（也称为散列值或散列地址）。
- 将结点按其关键字的散列地址存储到散列表中的过程称为散列（Hashing）。

1．常见的散列函数

- 除余法：令p是接近M的质数，设查找码为key，要求的桶号为T，计算T的散列函数为T=key %。
- 基数转换法：把查找码看作某个基数制上的整数，然后将它转换成另一基数制上的数。
- 平方取中法：先通过求关键字的平方值扩大相近数的差别，然后根据表长度取中间的几位数作为散列函数值。又因为一个乘积的中间几位数和乘数的每一位都相关，所以由此产生的散列地址较为均匀。
- 折叠法：此方法将关键字分割成位数相同的几部分（最后一部分的倍数可以不同），然后取这几部分的叠加和（舍去进位）作为哈希地址。如果关键字位数很多，而且关键字中每一位上数字分布大致均匀时，可以采用折叠法得到哈希地址。
- 随机数法：选择一个随机函数，取关键字的随机函数值为它的散列地址，即 h(key)=random(key)，其中random为伪随机函数，但要保证函数值在0到m–1之间。

2．冲突的解决

两个不同的关键字，由于散列函数值相同，因而被映射到同一表位置上。这种现象称为冲突或碰撞。发生冲突的两个关键字称为该散列函数的同义词。

冲突的频繁程度除与 h 相关外，还与表的填满程度相关。设 m 和 n 分别表示表长和表中填入的结点数，则将 $\alpha=n/m$ 定义为散列表的装填因子。α 越大，表越满，冲突的机会也越大，通常取 $\alpha \leqslant 1$。

解决冲突的方法是设法在散列表中找一个空位，通常有两类方法处理冲突，分别是开放定址法和拉链法。前者是将所有结点均存放在散列表 $T[0，…，m-1]$中，后者通常是将互为同义词的结点链成一个单链表，而将此链表的头指针放在散列表 $T[0，…，m-1]$中。

第 2 章

程序语言基础知识

程序语言是表达编程思想、描述计算过程的规范性语言。一般来说，程序语言可以分为低级语言和高级语言两大类。低级语言通常也称为面向机器语言。

自 1946 年现代电子计算机发明至今，计算机已经发展了将近 60 年，但是计算机依然只能理解自己的语言——机器指令。机器语言通过一系列的 0、1 字符表示命令和数据，难于记忆，编制出来的程序可读性很差，并且难于修改和维护。为了提高效率，人们开始用易于帮助记忆的符号来表示命令和数据，例如，使用 ADD 表示加，SUB 表示减，JMP 表示跳转等，这就是汇编语言。由于使用了助记符，汇编语言相对于机器语言来说比较容易记忆，用户编制程序的效率和程序的可读性、可维护性都得到了提高。但是汇编语言和机器语言十分接近，都是低级语言，与特定的计算机系统相关。使用机器语言或汇编语言进行程序设计均需要对特定的计算机系统有较深入的了解。

到现在，高级语言已经在程序设计的所有实质性领域里取代了机器语言和汇编语言，因为高级语言为程序员提供了与自然语言更接近、更熟悉的可读的记法形式，并与特定的机器无关，解除了面向机器的低级语言对程序员抽象思想的束缚。此外，高级语言带来了更具可用性的程序库和对错误检查的帮助。高级语言中存在着许多不同的程序设计范型，包括命令式程序设计（如 Pascal、C）、函数式程序设计（如 Lisp）、面向对象程序设计（如 C++、Java、Smalltalk）、逻辑程序设计（如 Prolog），以及面向主题程序设计（如 Aspect）等。

到目前为止，计算机都只能理解和执行机器语言，因此需要一种特殊的程序使计算机能够理解使用某一程序设计语言书写的程序，这种特殊的程序就是语言处理程序。语言处理程序可以分为两大类，分别是翻译程序和解释程序。

翻译程序的工作方式是把程序设计语言降低到机器水平，即把某一程序设计语言所写的程序（称为源程序）翻译成机器语言程序（称为目标程序），然后由计算机直接执行目标程序。当源程序语言为汇编语言时，翻译程序通常称为汇编程序，当源程序语言为高级语言时，翻译程序通常称为编译程序，图 2-1（a）说明了这种方式。而解释程序的工作方式是把程序设计语言看作解释器本身的语言。解释器的行为就像是一个能够直接运行某种程序设计语言的高级机器。解释器运行时同时取得程序和输入数据，遇到程序中的什么操作就进行相应的操作，并在需要时进行输入和输出，图 2-1（b）说明了这种方式。通常来说，翻译方式的程序执行效率比解释方式的要高，而另一方面，解释方式的灵活性要比翻译方式高。

前面所说的是纯粹的翻译和解释，而事实上这两种方法是可以互相结合运用的，如 Java 源程序就是先通过编译程序编译为以 Java 虚拟机的语言——BitCode 表示的程序，然后通过不同平台上的 Java 虚拟机解释执行。

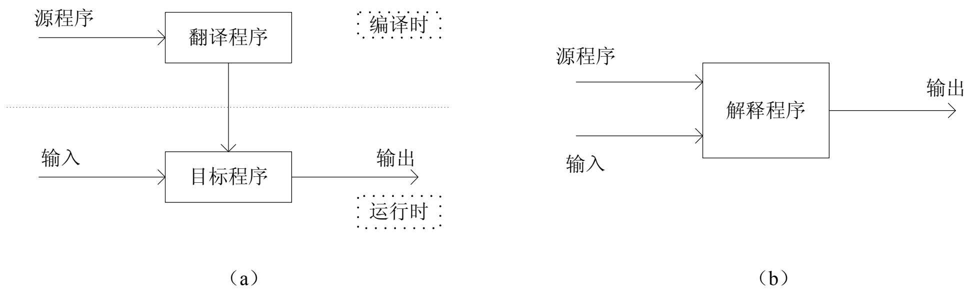


图 2-1　编译程序的工作方式

2.1　汇编系统基本原理

本节将介绍汇编系统基本原理。

2.1.1　机器语言与汇编语言

每一种特定型号的计算机系统都有自己特定的机器指令集合，集合中每条指令都代表一项具体的操作，例如，从内存取数据到寄存器。这个机器指令集合就是机器语言，由机器语言编写的程序就称为机器程序。机器指令本质上是一个特定长度的二进制串，特定的位表示操作码，而另外的位表示操作数。

由于机器程序都是由二进制的机器指令组成的，在编写机器程序时，不仅要记住特定操作码的二进制表示，还需要记下各个数据的地址的二进制表示。这非常不方便，而且容易出错，程序也很难读懂。于是人们就开始使用助记符（汇编指令）代表机器指令的操作码，并且使用伪指令（即不对应任何机器指令，只用于助记）和标号帮助确定数据或代码的位置，这就是汇编语言。由于汇编指令和机器指令是相对应的，所以每种特定型号的计算机系统都有自己的汇编指令集合。

由汇编指令编写的程序就是汇编程序，计算机不能直接执行汇编程序，而必须由一个特殊程序根据伪指令的控制把汇编程序转化为对应的机器语言程序。这个特殊的程序就是汇编程序。

2.1.2　汇编程序

如前面所述，汇编程序的基本工作包括：

- 将每一条可执行汇编指令转换成对应的机器指令。
- 处理源程序中出现的伪指令。

这一工作通常需要对汇编程序进行超过一次的扫描。

前面的分析已经指出，形成操作数地址的各个部分有可能出现符号，而符号是稍后语句的标号：

```
        SUB   1，C48
        ...
        C48  DC  48
```

为了计算各汇编语句中标号的地址，在汇编程序中设立单元地址计数器 LC，其初值一般为 0。以后每处理完一条可执行的汇编语句和与存储分配有关的伪指令（如定义常数语句、定义存储语句），LC 的值就增加相应长度，这样 LC 的值始终是下一个存储单元的相对地址。当处理一条汇编语句标号时，就将 LC 当时的值定义为标号值。由于符号使用可能出现在符号定义前，整个汇编程序工作要通过对源程序进行二次扫描才能完成。

第一次扫描的主要工作是定义符号的值。除设置单元计数器 LC 外，设立机器指令表 MOT1。由于本次扫描并不具体生成机器指令，MOT1 的每一元素只需两个域：机器指令记忆码和机器指令长度。在扫描过程中，将符号及其值记录在符号表 ST 中。此外，在第一次扫描中，还需要对与定义符号值有关的伪指令进行处理。为了叙述方便，不妨设立伪指令表 POT1，POT1 表的每一个元素只有两个域：伪指令记忆码和相应处理子程序入口。对第一次扫描的描述如下。

①单元计数器 LC 置初值 0。

②打开源程序文件。

③反复执行如下操作。

- 从源程序文件读下一条语句。
- 如果该语句有标号，则将标号和LC当时值送符号表ST。
- 根据语句操作码，执行如下操作。
  - ➢ 如果是可执行汇编语句，K是查MOT1表所得机器指令长度，则LC：=LC+K。
  - ➢ 如果是伪指令记忆码，则调用POT1表相应元素所规定的子程序。
  - ➢ 如果是非法记忆码，则调用出错子程序。

直至语句操作码是 END 为止。

④关闭源程序文件。

第二次扫描的目的是产生目标程序。除前一次扫描所生成的符号表 ST 外，需要建立机器指令表 MOT2，该元素包含下面区域：机器指令记忆码，机器指令的二进制操作码（binary_code）、格式指示（type）和长度（length）。还设立第二次扫描的伪指令表 POT2，它的每一元素仍是两个区域：伪指令记忆码和相应处理子程序入口。所不同的是，在第二次扫描中，伪指令有着完全不同的处理。

在第二次扫描中，可执行汇编语句应被翻译成对应的二进制代码机器指令。这一工作涉及两个方面：把机器指令记忆码转换成二进制机器指令操作码，以及求出操作数区各操作数的值（用二进制数表示）。在此基础上，可以装配出二进制代码的机器指令。对于第一部分工作，只要根据机器指令记忆码查机器指令表 MOT2，就可以获得相应二进制数表示的机器指令操作码。从求值的角度来说，第二部分工作并不复杂。由于形成内存操作数地址的各个部分都以表达式的形式出现，因此统一定义一个过程 eval-expr（index，value）。调用时，只要将表达式在汇编语句缓冲区 S 开始位置通过 index 传递给此过程，该过程就通过 value 返回此表达式的值。例如，虚拟计算机 COMET 的机器指令可归属于“X”型指令，其汇编语句如下。

```
OP  R1，N2，X2
OP  R1，N2
```

可以写出下面处理“X”型指令的程序段（假定 index 已指向操作数在缓冲区 S 的首址）：

```
eval_expr（index，R1）;
index：=index+1;
eval_expr（index，N2）;
if S [index]='，'then
  begin
    index：=index+1;
    eval_expr（index，X2）
  end
else
  X2：=0;
```

其他类型指令的处理操作数的程序段都可以类似地写出。设当前可执行汇编语句的操作记忆码在 MOT2 表的索引值为 i，则整个可执行汇编语句的处理可以描述如下：

```
OP：=MOT2 [i].binary_code;
TYPE：=MOT2 [i].type;
case TYPE of
  'X'：求 X 型指令操作数各个部分值，然后按规定字节形成指令;
  ...
end;
  将形成指令送往输出区;
```

在第二次扫描中，DS 伪指令的主要目的是保留存储空间。不妨设立一个工作单元 *k*，用于累计以字节为单位的存储空间大小，*k* 初值为 0。从 DS 伪指令的操作数区求出 *k* 的大小后，就向输出区送 *k* 个空格以达到保留所规定存储单元的目的。DC 伪指令处理和 DS 伪指令类似，只不过向输出区送的是所转换得到的常量。最后，START 伪指令工作可能是输出目标程序开始的标准信息，而 END 伪指令则可能是输出目标程序结束的标准信息，这些信息都是为装配程序提供的。

2.2 编译系统基本原理

本节将介绍编译系统基本原理。

2.2.1 编译概述

编译程序的职能是把使用某程序设计语言书写的程序翻译为等价的机器语言程序，所谓等价是指目标程序执行源程序的预定任务。一般来说，编译程序分为以下几个部分：词法分析，语法分析和语义分析，代码优化，代码生成，以及符号表管理。各部分之间的关系如图 2-2 所示。

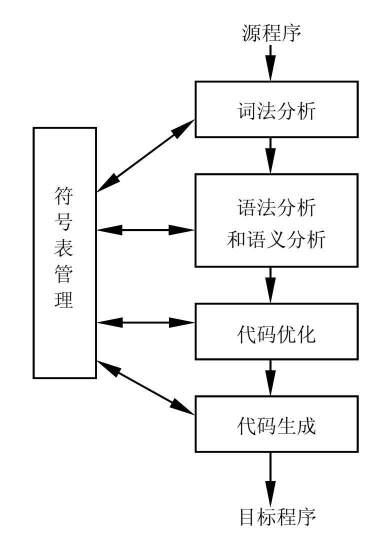


图 2-2 编译程序结构框图

词法分析程序是编译程序的第一个部分，它的输入是源程序中由字符组成的符号。编译程序需要把程序的

这种外部形式转换成适合后续程序处理的形式，其功能如下。

- 识别出源程序中意义独立的最小词法单位——单词，并且确定其类型（例如是标识符、关键字、操作符还是数字等）。
- 删除无用的空格、回车和其他与输入介质有关的无用符号，以及程序注释。
- 报告分析时的错误。

经过词法分析程序处理后，源程序就转化为单词串。每个单词都是一个意义独立的单位，其所包含的信息量个数固定。语法分析程序根据特定程序设计语言的文法规则，检查单词串是否符合这些规则。一旦语法分析程序分解出其中一个文法结构，该结果的语义分析程序就进行相应的语义检查，在有需要的时候输出相应的中间代码。这里的中间代码可以理解为假想的虚拟机的指令，其执行次序反映了源程序的原始定义。语法和语义分析程序是编译程序中的关键部分。

中间代码作为代码生成程序的输入，由代码生成程序生成特定的计算机系统下的机器代码。为了提高目标代码的运行效率和减小目标代码大小，也可以在语法语义分析程序与代码生成程序之间插入代码优化程序。代码优化程序在不改变代码所完成的工作的前提下对中间代码进行改动，使其变成一种更有效的形式。

编译程序在完成其任务的过程中，还需要进行符号表的管理和出错处理。在符号表中登记了源程序中出现的每一个标识符及其属性。在整个编译过程中，各部分程序都可以访问某标识符的属性，包括标识符被说明的类型、数组维数、所需存储单元数，所分配的内存单元地址等。错误管理程序是在分析程序发现源程序有错误而无法继续工作时进行其工作的。其任务是记录并向用户报告错误及其类型和位置，或者尝试进行某种恢复工作。

2.2.2 形式语言基本知识

首先介绍关于字母表和符号串的定义。

无论是自然语言还是形式语言，均是由特定的符号，如字母、数字等组合而成的，符号的非空有限集合称为字母表。由某一字母表中的符号组成的有限符号序列称为该字母表的符号串。符号串 α 的长度是指 α 中出现的符号个数，记为 $|\alpha|$。空串的长度为 0，用 ε 表示。

符号串 α 的前缀是指 α 的末尾删除零个或多个符号后得到的符号串，如 pro 是 program 的一个前缀。符号串 α 的后缀是指 α 的开头删除 0 个或多个符号后得到的符号串，如 gram 是符号串 program 的一个后缀。符号串 α 的子串是删除了 α 的前缀和后缀后得到的符号串，如 og 是 program 的子串，α 的前缀和后缀都是它的子串。对于任意符号串 α，其自身和 ε 都是 α 的前缀、后缀，也是 α 的子串。符号串 α 的真前缀、真后缀和真子串是指除空串 ε 和 α 自身外，α 的前缀、后缀和子串。

符号串 α 的子序列是从 α 删除 0 个或多个符号（这些符号不要求是连续的）而得到的符号串。

下面介绍符号串之间的运算。

符号串 α、β 的连接 $\alpha\beta$ 是指把 β 写在 α 的后面得到的符号串，从空串的定义可以推出 $\varepsilon\alpha=\alpha\varepsilon=\alpha$。符号串 α 的方幂 α^n 定义为 $\alpha\alpha\cdots\alpha$（n 个），由 $\alpha^0=\varepsilon$，$\alpha^1=\alpha$。

术语“语言”表示某个确定的字母表上符号串的任何集合。空集合{ }和只包含空串的集合{ε}也是符合定义的语言。在字符串运算的基础上，可以定义语言的运算：

①语言 L 和 M 的合并，$L \cup M = \{s \mid s \in L 或 s \in M\}$

②语言 L 和 M 的连接，$LM = \{st \mid s \in L, t \in M\}$

③语言 L 的 Kleene 闭包，$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cdots$

④语言 L 的正闭包，$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup L^3 \cdots$

上述对语言的定义是非形式化的，下面要介绍形式化的语言定义，这里首先引入文法的概念。

所谓文法 G 是一个四元组，$G = \{V_T, V_N, S, P\}$。其中 V_T 是一个非空有限的符号集合，它的每个元素成为终结符号。V_N 也是一个非空有限的符号集合，它的每个元素称为非终结符号，并且有 $V_T \cap V_N = \Phi$。$S \in V_N$，称为文法 G 的开始符号。P 是一个非空有限集合，它的元素称为产生式。所谓产生式，其形式为 $\alpha \rightarrow \beta$，α 称为产生式的左部，β 称为产生式的右部，符号“→”表示“定义为”，并且 α、$\beta \in (V_T \cup V_N)^*$，$\alpha \neq \varepsilon$，即 α、β 是由终结符和非终结符组成的符号串。开始符 S 必须至少在某一产生式的左部出现一次。另外可以对形如 $\alpha \rightarrow \beta$，$\alpha \rightarrow \gamma$ 的产生式缩写为 $\alpha \rightarrow \beta \mid \gamma$，以方便书写。

1956 年，著名的语言学家 Noam Chomsky 首先对形式语言进行了描述，把文法定义为四元组，并且根据对产生式所施加的限制的不同，把文法分成了 4 类，并定义了相应的 4 类形式语言。表 2-1 所示为 4 类文法及其产生的语言。

表 2-1　4 类文法及其产生的语言

文 法 类 型	产生式的限制	文法产生的语言
0 型文法	$\alpha \rightarrow \beta$ 其中 α、$\beta \in (V_T \cup V_N)^*$，$\mid \alpha \mid \neq 0$	0 型语言
1 型文法	$\alpha \rightarrow \beta$ 其中 α、$\beta \in (V_N \cup V_T)^*$，但需$\mid \alpha \mid \leqslant \mid \beta \mid$	1 型语言，即上下文有关语言
2 型文法	$A \rightarrow \beta$ 其中 $A \in V_N$，$\beta \in (V_N \cup V_T)^*$	2 型语言，即上下文无关语言
3 型文法	$A \rightarrow \alpha \mid \alpha B$（右线性）或 $A \rightarrow \alpha \mid B\alpha$（左线性） 其中，$A, B \in V_N$，$\alpha \in V_T \cup \{\varepsilon\}$	3 型语言，即正规语言，又分为左线性语言和右线性语言

对于文法 G[S]，称 αAβ 直接推导出 αγβ（也可以说 αγβ 是 αAβ 的直接推导），仅当 $A \rightarrow \gamma$ 是文法 G 的一个产生式，且 α，$\beta \in (V_T \cup V_N)^*$，记作 $\alpha A \beta \Rightarrow \alpha \gamma \beta$。如果存在直接推导序列：$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$，则称该序列为 α_0 到 α_n 的长度为 n 的推导，也称为 α_0 可以推导出 α_n，记作 $\alpha_0 \overset{+}{\Rightarrow} \alpha_n$。如果 n=0，即 $\alpha_0 = \alpha_n$ 或 $\alpha_0 \overset{+}{\Rightarrow} \alpha_n$，则记为 $\alpha_0 \overset{*}{\Rightarrow} \alpha_n$。如果在每一步的直接推导中，都对最左边的非终结符应用相应的产生式的右部来代替，则

称这种推导为最左推导。类似的，如果在每一步的直接推导中，都对最右边的非终结符应用相应的产生式的右部来代替，则称这种推导为最右推导。

在文法G[S]中，如果存在 $S \overset{*}{\Rightarrow} \alpha$，则称α是文法G的一个句型，仅含终结符号的句型是文法G的一个句子。语言L(G)是由文法G产生的所有句子组成的集合，其形式定义为：$L(G)=\{\alpha \mid S \overset{+}{\Rightarrow} \alpha \text{ 且 } \alpha \in V_T^*\}$。称文法G1和文法G2是等价的，如果有L(G1)=L(G2)，即有可能不同的文法产生相同的语言。

对于文法G，如果 $S \overset{*}{\Rightarrow} \alpha A \delta$ 且 $A \overset{+}{\Rightarrow} \beta$，则称β为一个关于非终结符号A的、句型αβδ的短语。如果 $A \Rightarrow \beta$，则称β为直接短语。一个句型的最左直接短语称为该句型的句柄。

要检查符号串x是否是文法G的一个句型或者句子，就要检查是否存在一个由S到α的x的推导。推导树的每一个结点和终结符或者非终结符相关联。和终结符关联的结点是叶结点，而与非终结符相关联的结点可以是叶结点，也可以是非叶结点，树的根结点为文法的开始符号S。已知符号串x在文法G中的一个推导，就可以构造相应的推导树。将x中的每一步产生式的应用表达从所替代的非终结符号生长出新的树杈，且子结点自左向右逐个和产生式的右部符号相关联。因此，每棵推导树的终端结点自左至右所构成的字符串应该是文法G的一个句型，如果所有的终端结点都是与终结符关联的，则该字符串是文法G的一个句子，此时该推导树是完全推导树。考查文法G=（{a，b}, {S，A}，S，P），其中：

$$S \rightarrow aAS|a$$

$$A \rightarrow SbA|SS|ba$$

句型aabAa相对应的推导树构造的全过程如图2-3所示。

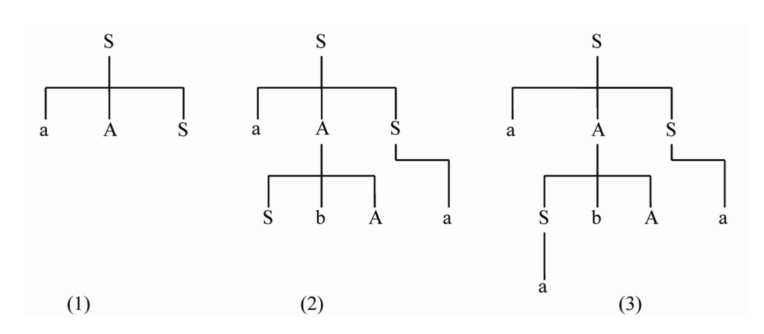


图2-3　句型aabAa相对应的推导树构造的全过程

这里再引入子树的概念。分析树的子树是树中一个特有的结点连同它的全部后裔和连接这些后裔的边，因此子树的根结点可能不是开始符号。子树和短语的关系十分密切。一棵树的所有叶子自左至右所构成的字符串就是相对于子树根的短语，一个句型的分析树中最左那棵只有父子两代的子树的所有叶子由左至右所构成的字符串就是该句型的句柄。

如果一文法的句子存在两棵不同的分析树，则称该句子为二义性的；如果一文法包含二义性的句子，则称该文法为二义性的，否则该文法是无二义性的。需要注意的是，文法的二义性和语言的二义性是不同的。可能出现的情况是有两个文法 G 和 G'，且 G 有二义性而 G'无二义性，但 L(G)=L(G')，即文法 G 与文法 G'产生相同的语言。因此，有时可以在不改变一个二义性文法的句子集合的情况下改变该文法，得到一个无二义性的文法。但是，也有一些语言，它们不存在无二义性的文法，这样的语言称为先天二义性的语言。

2.2.3 词法分析

词法分析是整个分析过程的一个子任务，它把构成源程序的字符串转换成语义上关联的单词符号（包括关键字、标识符、常数、运算符和分界符等）的序列。词法分析可以借助于有限自动机的理论与方法进行有效的处理。

1. 有限状态自动机

有限状态自动机是具有离散输入和输出的系统的一种数学模型。系统可以处于内部状态的任何一个之中，系统当前状态概括了有关过去输入的信息，这些信息对在后来的输入上确定系统的行为是必需的。有限状态自动机与词法分析程序的设计有着密切的关系。下面是确定的有限状态自动机的形式定义：

一个确定的有限状态自动机 M（记作 DFA M）是一个五元组：

$$M=(\Sigma,\ Q,\ q_0,\ F,\ \delta)$$

- Q是一个有限状态集合。
- Σ是一个字母表，其中的每个元素称为一个输入符号。
- $q_0 \in Q$,称为初始状态。
- $F \subseteq Q$，称为终结状态集合。
- δ是一个从$Q\times\Sigma$（Q与Σ的笛卡儿乘积）到Q的单值映射：

$$\delta(q,\ a)=q' \qquad (q,\ q' \in Q,\ a\in\Sigma)$$

表示当前状态为 q，输入符号为 a 时，自动机将转换到下一个状态 q'，q'称为 q 的一个后继。

若$Q=\{q_1,\ q_2,\cdots,\ q_n\}$，$\Sigma=\{a_1,\ a_2,\ \cdots,\ a_n\}$，则$(\delta(q_i,\ a_j))n\times m$ 是一个 n 行 m 列矩阵，称为 DFA M 的状态转换矩阵，或称转换表。

有限状态自动机可以形象地用状态转换图表示，设有限状态自动机：

$$\text{DFA M}=(\{S,\ A,\ B,\ C,\ f\},\ \{1,\ 0\},\ S,\ \{f\},\ \delta),$$

其中：

$$\delta(S,\ 0)=B,\quad \delta(S,\ 1)=A,\quad \delta(A,\ 0)=f,\quad \delta(A,\ 1)=C,\quad \delta(B,\ 0)=C,\quad \delta(B,\ 1)=f,$$
$$\delta(C,\ 0)=f,\quad \delta(C,\ 1)=f$$

其对应的状态转换图如图 2-4 所示。

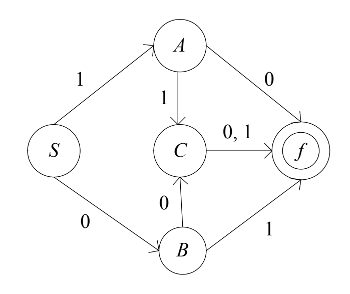


图 2-4　状态转换图

图 2-4 中的圈表示状态结点，其中双圈表示终结状态结点。而边表示状态的转换，代表映射。边上的符号表示此转换需要输入的符号，代表映射的输入。

对于 Σ 上的任何字符串 $w \in \Sigma^*$，若存在一条从初态结点到终态结点的路径，在这条路径上的所有边的符号连接成的符号串恰好是 w，则 w 被 DFA M 所识别（或接受、读出）。DFA M 所能识别的符号串的全体记为 $L(M)$，称为 DFA M 所识别的语言。如果对所有 $w \in \Sigma^*$，以下述的递归方式扩张 δ 的定义：

$$\delta(q，\varepsilon)=q$$

$$\delta(q，wa)=\delta(\delta(q，w)，a)，对任何\ a\in\Sigma，q\in Q$$

则可以把 DFA M 所识别的语言形式定义为：

$$L(M)=\{w \mid w \in \Sigma^*，若存在 q \in F，使\ \delta\ (q_0,w)=q$$

前面介绍的是确定的有限自动机，即一个状态对于特定的输入字符有一个确定的后继状态。而当一个状态对于特定的输入字符有一个以上的后继状态时，称该有限自动机为非确定有限自动机（记作 NFA M），其形式定义如下。

一个非确定的有限自动机 M 是一个五元组：

$$M=(\Sigma，Q，q_0，F，\delta)$$

其中 Σ、Q、q_0、F 的意义和 DFA 的定义一样，而 δ 是一个从 $Q\times\Sigma$ 到 Q 的子集的映射，即 δ：$Q\times\Sigma\rightarrow 2^Q$，其中 2^Q 是 Q 的幂集，即 Q 的所有子集组成的集合。

与确定的有限自动机一样，非确定有限自动机同样可以用状态转换图表示，所不同的是，在图中一个状态结点可能有一条以上的边到达其他状态结点。同样，对于任何字符串 $w \in \Sigma^*$，若存在一条从初态结点到终态结点的路径，在这条路径上的所有边的符号连接成的符号串恰好是 w，则称 w 为 NFA M 所识别（或接受或读出）。若 $q_0\in F$，这时 q_0 既是初始状态，也是终结状态，因而有一条从初态结点到终态结点的 ε －路径，此时空符号串可以被 NFA M 接受。NFA M 所能识别的符号串的全体记为 $L(M)$，称为 NFA M 所识别的语言。

对任何一个 NFA M，都存在一个 DFA M’使 $L(M’)=L(M)$，这时称 $M’$ 与 M 等价。构造与 M 等价的 $M’$ 的基本方法是让 $M’$ 的状态对应于 M 的状态集合。即如果有 $\delta(q，a)=\{q_1，q_2,\cdots，q_n\}$，则把 $\{q_1，q_2,\cdots，q_n\}$ 看作 $M’$ 的一个状态，即 $M’$ 中的状态集合 Q'的一个元素。

对于一个非确定有限自动机，如果把 δ 扩展为从 $Q\times\Sigma\cup\{\varepsilon\}$ 到 2^Q 的映射，则称该自动机为带 ε －转移的非确定有限自动机。同样，对于带 ε －转移的非确定有限自动机，也可以构造与之等价的不带 ε －转移的非确定有限自动机。

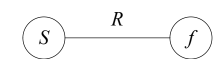
2．正规表达式

正规表达式是一个十分有用的概念，它紧凑地表达有限自动机所接受的语言。对正规表达式的递归定义为：一个正规表达式是按照一组定义规则由一些较简单的正规表达式所组成的。在字母表 Σ 上的正规表达式可以使用如下规则定义。

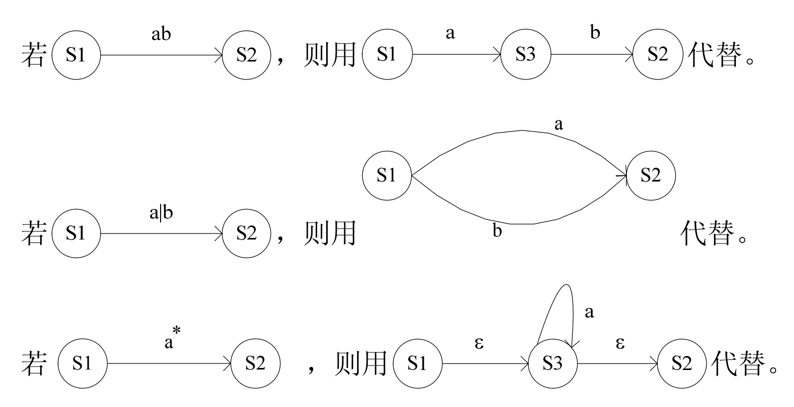
- ε 和 Φ 是 Σ 上的正规表达式，它们所表示的语言分别为{ ε }和 Φ。
- 如果 a 是 Σ 内的一个符号，则a是一个正规表达式，所表示的语言为{a}，即包含符号串 a 的集合。
- 如果 r 和 s 分别是表示语言 $L(r)$ 和 $L(s)$ 的正规表达式，那么：
  - $(r)|(s)$ 是一个表示(s)—R—(f)的正规表达式；
  - $(r)(s)$ 是一个表示 $L(r)L(s)$ 的正规表达式；
  - $(r)^*$ 是一个表示 $(L(r))^*$ 的正规表达式；
  - (r) 是一个表示 $L(r)$ 的正规表达式。

通常在正规表达式中，一元运算符“*”具有最高的优先级，连接运算具有次优先级，运算符“|”具有最低优先级，这三个运算都是左结合的。每一个正规表达式 R 都对应一个有限自动机 M，使 M 所接受的语言就是正规表达式的值。经过如下步骤可以从一个正规表达式 R 构造出相应的有限自动机 M。

首先定义初始状态 S 和终止状态 f，并且组成有向图：



然后反复应用如下规则：



直到所有的边都以 Σ 中的字母或 ε 标记为止。由此产生了一个带 ε －转移的非确定有限自动机，然后可以通过上述介绍的方法，把该自动机转换成确定有限状态自动机。

下面举一个例子说明自动机理论在词法分析程序中的应用。C 语言中对标识符的规定为由“_”或以字母开头的由“_”、字母和数字组成的字符串，该标识符的定义可以表示为如下正则表达式：

$$(_\,|\,a)(_\,|\,a\,|\,d)^*$$

式中的 α 代表字母字符{*A*，…，*Z*，*a*，…，*z*}，*d* 代表数字字符{0，1，…，9}。利用前面的方法构造出如图 2-5 所示的有限自动机。

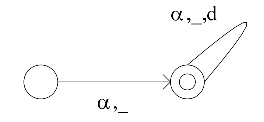


图 2-5　有限自动机图例

该自动机所接受的语言就是 C 语言中的标识符。

在有限自动机的状态转换过程中，需要执行相关的语义动作。例如，当识别到一个标识符时，需要在符号表中添加该标识符，并且向语法分析程序输送表示该标识符的单词。

2.2.4　语法分析

1．下推自动机

为了帮助理解语法分析程序，在这里先介绍下推自动机的概念。下推自动机（PDA）是自动机理论中定义的一种抽象的计算模型。下推自动机比有限状态自动机复杂，除有限状态组成部分外，还包括一个长度不受限制的栈；下推自动机的状态迁移不但要参考有限状态部分，也要参照栈当前的状态；状态迁移不但包括有限状态的变迁，还包括一个栈的出栈或入栈过程。下推自动机可以形象地理解为，把有限状态自动机扩展使之可以存取一个栈。

下推自动机存在确定与非确定两种形式，两者并不等价。

每一个下推自动机都接受一种形式语言，确定下推自动机接受的语言是上下文无关语言。如果把下推自动机扩展，允许一个有限状态自动机存取两个栈，则会得到一个能力更强的自动机，这个自动机与图灵机等价。

2．自顶向下语法分析

该分析方法是以文法开始符号为根，试图寻找以输入符号串为端点的推导树。每次都以最左边的那个非终结符为根，选择适当的产生式，往下扩展子树。

考虑上下文无关文法 *G*=({*a*，*b*，*c*}，{*S*，*B*}，*S*，*P*)，其中：

- $S \rightarrow aBS$
- $S \rightarrow b$
- $B \rightarrow cBS$
- $B \rightarrow d$

设输入字符串为 *w*=*acdbb*，如果 *w* 是 *G* 的一个句子，则可以从 S 出发构造一棵以 *w* 为端点的推导树。首先构造分析该文法的下推自动机，对于输入串 *acbb*，则下推自动机生成表如表 2-2 所示。

表 2-2　下推自动机生成表

下　推　栈	输　　入	文法产生式
Z_0S	acdbb$	S→aBS
Z_0SB	cdbb$	B→cBS
Z_0SSB	dbb$	B→d
Z_0SS	bb$	S→b
Z_0S	b$	S→b
Z_0	------	------

这里的关键是，每个产生式都有一个由相应终结符组成的选择集合。只要产生式左部相同的选择集合两两不相交，则下推自动机就能确定性地工作。

3．自底向上语法分析

自底向上分析技术是从输入符号串出发，试图把它归约为识别符号。从语法树的角度看，这个技术首先以输入符号作为语法树的末端结点，然后向根结点方向构造语法树。

2.2.5　语法翻译

虽然编译程序可以直接把一个源程序翻译成目标程序，但是在许多编译系统的设计中，仍采用独立于机器的中间代码作为过渡。其优点是便于编译系统的建立和移植，并且便于进行独立于机器的代码优化。常见的中间代码表示有语法树、后缀式和三地址代码。这里介绍后缀式和三地址代码表示。

后缀表示又称为逆波兰表示，最初是用于表示表达式的计算次序。在后缀表示中，运算符紧跟在相应的运算对象后。例如，表达式$(A+B)*C$，使用后缀表示为 $AB+C*$。在后缀表示中，运算符既表示了对运算对象所执行的运算，也表示了这一运算之前的中间结果。因此，在后缀表示中不需要使用括号。

通过使用栈，计算后缀表达式的算法如下。

- 如果P的下一项是运算对象，则将它压入栈。
- 如果P的下一项是二元操作符（即需要参数个数为2的运算符），则对栈顶两个对象实施运算，并且将运算的结果代替这两个运算对象而进栈。
- 如果P的下一项是一元操作符（即需要参数个数为2的运算符），则对栈顶对象实施运算，并且将运算的结果代替这个对象而进栈。
- 如果P的下一项是n元操作符（即需要参数个数为n的运算符），那么它的参数就是栈顶的n项，把该运算符作用于这n项，得到的结果作为操作数替代栈顶的n项。
- 最后的结果留在栈顶。

例如，$AB+C*$的计算过程为：

①A 进入堆栈。

②B 进入堆栈。

③遇到二元运算符“+”，则 AB 出堆栈，并将 $A+B$ 的结果 X 送入堆栈。

④C 进入堆栈。

⑤遇到二元运算符“*”，则 CX 出堆栈，并将 $C*X$ 的结果送入堆栈。

⑥现在堆栈顶部存放的是整个表达式的值。

把运算符扩展到 n 元后，后缀表达式可以表示为：$O_1O_2 \ldots O_n\ \theta$，其中 θ 是 n 运算符，运算对象的个数由运算符决定。

2.2.6 代码生成

代码生成时编译程序的最后一个阶段，它将程序的中间代码表示作为输入，并产生等价的目标代码作为输出。

类似于中间代码，目标代码也有若干种形式：绝对机器代码、可再定位机器语言、汇编语言等。为了讨论方便，假定代码生成程序产生用汇编语言书写的目标程序。

为此，必须对目标机器及其指令系统做出定义。这里，将采用具有多个通用寄存器的机器作为目标机器。我们的目标机器是一个按字节编址的机器，以 4 字节为一个字，有 n 个通用寄存器 R_0，R_1，…，R_{n-1}，并有形如 op src、dest 的两地址指令。其中 op 为操作码，src 和 dest 称为源和目标，是数据域。此目标机器有如下的基本指令：

- MOV（将源移到目标中）。
- ADD（将源加到目标中）。
- SUB（在目标中减去源）。

源和目标域并不足以用来存放存储地址，因此一条指令的源和目标是通过将寄存器与带有地址方式的存储单元结合起来确定的。下面的 contents(a)，表示由 a 所代表的存储单元或寄存器的内容。表 2-3 所示为各地址方式及其在汇编语言中的形式。

表 2-3　各地址方式及在汇编语言中的形式

地 址 方 式	汇 编 形 式	地　　址
直接地址方式	M	M
寄存器方式	R	R
间接寄存器方式	*R	contents(R)
索引方式	c(R)	c+contents(R)
间接索引方式	*c(R)	contents(c+contents(R))
字面常数	#C	C

寄存器分配是代码生成中的一个重要问题。寄存器是有限的资源，并且用途广泛。这些寄存器是否有效，直接涉及目标代码质量的好坏。

首先介绍基本块的概念。一个基本块是这样一个连续语句序列：其中控制流从第一条语句（称为入口语句）进入，从最后一条语句离开，没有中途停止或分支。划分三地址程序基本块的方法如下。

①首先确定基本块的入口。

- 代码序列的第一条语句是一条入口语句。

- 任何一个条件或无条件转移语句转移到的那条语句是一条入口语句。
- 任何紧接在一个条件或无条件转移语句后面的语句是一条入口语句。

②对于上述求出的每一个入口语句，其基本块由该语句到下一条入口语句（不包括这一条入口语句），或到一条转移语句（包括转移语句），或到一个停止语句（包括停止语句）之间的语句序列组成。

为了有效利用寄存器，首先设立一个寄存器工作表 RVALUE，表中的每个单元对应一个寄存器，用以记录运行时刻寄存器中值的情况。开始时，各寄存器的值为空。当向一个寄存器中存放变量时，只要将变量挂在对应的 RVALUE 单元上。这样，RVALUE 中的单元值就反映了运行时寄存器的值。每当需要将某变量的值取至寄存器时，代码生成程序首先检查寄存器中是否有该值，如果有，则不必生成相应的取数指令。

除了关心寄存器的值，在此还要关心寄存器的状态，一般状态会分为如下几种情况。

- 寄存器不含有任何值，即该寄存器处于空闲状态。
- 寄存器中的值是程序中某变量的值，但与正在处理的中间代码基本块中的后续语句无关。
- 寄存器中的值是与正在处理的中间代码无关的中间变量的值。
- 寄存器中的值是正在处理的中间代码基本块中后续语句需要引用的。
- 寄存器中的值是当前要处理的中间代码的某操作数的值。

上述状态对寄存器的分配有至关重要的意义，为此，设立一张寄存器状态表 RSTATE，表中每个单元对应一个寄存器，它的值反映寄存器的状态。代码生成程序应该先选择状态为 1 的寄存器进行分配，其次是状态为 2 的寄存器，再次是状态为 3 的寄存器，依次类推。这样可以避免生成不必要的存取指令。

至此，所谓的分配寄存器，就是往某一约定工作单元 j 中送一个该寄存器的号码，并将 RVALUE 中对应的单元内容送到工作单元 jVALUE。代码生成程序在处理每一条三地址代码前，必须根据代码各分量及寄存器当时的值之间的关系修改 RSTATE。在处理完后，必须及时修正 RSTATE 和 RVALUE 的值，以正确反映寄存器的值和状态。

由于机器指令的位移区的位数是有限的，因此与内存有关的指令只能访问有限范围内的存储单元。要存取超过此范围的存储单元，就需要使用前面介绍的地址方式，把特定的寄存器定义为变址寄存器。通过设定变址寄存器的值，并运用各种地址方式，则可以灵活地访问各存储单元。

访问变量的过程为，①按照程序语言的作用域规则，沿活动记录的访问链定位该变量所在的活动记录；②把变址寄存器定位为该活动记录的相应存储工作区的首地址；③使用该变量的偏移和变址寄存器的值访问该存储单元。这里的关键是要准确定位变量所在的块及其首地址。

最后，介绍一下简单的代码生成的算法。代码生成程序的工作过程为逐个处理三地址代码，在每次处理一条三地址代码前，将该三地址代码放在固定的地方，然后根据三地址代码的操作码转入相应的处理程序，直至所有的三地址代码全部处理完毕。这样，代码生成程序可以简单表达为输入三地址代码和处理三地址代码。

根据上述预备，可以写出将变量赋值 a:=b 的代码的生成算法：

- 检查RVALUE表，*b*的值是否在寄存器中。
- 如果*b*的值不在寄存器中，则：
  - ➢ 即为其分配一个寄存器R1。
  - ➢ 定位变量*b*的地址，设置变址寄存器*R*d，使其指向*b*所在块的首地址A，输出语句MOV Rd, A。
  - ➢ 根据*b*的偏移*D*，把*b*挂在*R*1对应的RVALUE单元中，输出MOV *R*1, **D*(*Rd*)；
- 如果b的值在寄存器中，为描述方便也假定其为*R*1。
- 定位变量*a*的地址，设置变址寄存器*Rd*，使其指向*a*所在块的首地址A，输出语句MOV *Rd*, A。
- 根据*a*的偏移*D*，把*b*挂在*R*1对应的RVALUE单元中，输出MOV **D*(*Rd*), *R*1。

其他的语句也可以使用类似的过程生成目标代码。

2.3 程序语言的控制结构

程序设计语言的控制结构提供了一个将操作和数据组合成程序和程序组的基本框架，考虑的是如何组织数据和操作，使其成为一个可执行的程序，这包括两个方面的内容，一是对操作执行次序的控制，称其为顺序控制；二是对程序中的过程间数据传递的控制，称其为数据控制。

控制结构可以简单地分为 3 类。

- 表达式是程序语句的基本组成，体现了程序控制和数据改变的方法。
- 用在语句间或一组语句中的结构，如条件语句和循环语句。
- 过程结构，如过程调用和协同程序。

2.3.1 表达式

1．表达式的前缀、后缀、中缀表示法

1）*前缀表示法*

在前缀表示法中，以从左到右的顺序先写操作符后写操作数，如果操作数本身是一个具有操作数的操作，则对其使用同样的规则。例如，公式(a+b)(a−b)，使用前缀表达式表示将变为$* + a\ b - a\ b$。函数调用可以看作一种前缀表达式，因为一般是把操作符函数名写在它的参数的左边，像$f(a, b)$这样。

以前缀表示的表达式可以在一次扫描后计算出值，前提是要明确知道每个操作符的参数的数目。用如下的算法通过使用一个执行栈，计算给定的前缀表达式 *P*。

如果 *P* 的下一项是一个操作符，将它压入栈，并把参数计数器设置为该操作符所需要的参数的数目 *n*。

- 如果*P*的下一项是一个操作数，把它压入栈。
- 如果栈顶操作数的个数为*n*，则把操作符作用于这*n*个操作数，得出的结果替换该操作符和它所有的参数，作为操作数压入栈。

前缀表达式的计算方法意味着在每个操作数压入栈后都必须检查操作数的数目是否满足最近栈顶的操作符的要求，而后缀表达式就无须做这种检查。

2）后缀表示法

后缀表示法类似于前缀表示法，不同之处在于操作符跟在操作数之后。前面的公式使用后缀表示法时表示为 $a\ b + a\ b - \ *$ 。由于这一点不同，在计算后缀表达式时，当扫描到操作符时，栈中已压入了它的操作数。因此计算后缀表达式的算法如下。

- 如果P的下一项是操作数，将它压入栈。
- 如果P的下一项是n元操作符（即需要参数个数为n的操作符），那么它的参数就是栈顶的n项，把该操作符作用于这n项，得到的结果作为操作数替代栈顶的n项。

由于后缀表达式的计算是直接的，并且易于实现，因此它是很多翻译器产生表达式代码的基础。

3）中缀表示法

中缀表示法是日常最通用的用法，但是在程序语言中使用中缀表示法会产生下面的问题。

- 由于中缀表示法仅适合于二元操作符，一种语言不能只使用中缀表示法，而必须结合中缀与前缀表示法。这种混合使用会使翻译过程相对复杂。
- 当一个以上的中缀操作符出现在表达式中时，如果不使用括号就有可能产生二义性。

考虑表达式 $10-2\times5$ 的值，我们会认为其值是 0，而这仅仅是因为我们已经习惯把一个隐含规则应用于表达式的求值，这就是先乘除后加减。若把规则定义为先加减后乘除，则表达式值应为 40。括号可以消除任何表达式的二义性，但在复杂的表达式中，将会导致深层次的括号嵌套而产生混乱。因此，程序语言通常都引入隐含的控制规则，使得大多数的括号的使用成为不必要，这就是操作符的优先级和结合性。

2．操作符的优先级和结合性

操作符的优先规则是指可以出现在表达式中的操作符的优先次序，操作符在该次序中的级别就是该操作符的优先级。在有处于一个以上优先级的操作符的表达式中，具有较高优先级的操作符先执行。

结合性规定了相同等级的多个操作符的操作次序。例如，在 $a-b-c$ 中，应是第一个减法还是第二个减法先完成呢？通常的隐含规则是从左到右的结合。因此，$a-b-c$ 被看作$(a-b)-c$。表 2-4 所示为 C 语言中的操作符的优先级和结合性。

表 2-4　C 语言中的操作符的优先级和结合性

优　先　级	操　作　符	操 作 符 名	结　合　性
17	Tokens, a[k], f()	文字，下标，函数调用	左
	., ->	选择	左
16	++, --	后缀增量/减量	左
15	++, --	前缀增量/减量	左
	~, -, sizeof	一元操作符，存储量	左
	!, &, *	逻辑非，取地址，指针	右

续表

优 先 级	操 作 符	操 作 符 名	结 合 性
14	（类型名）	强制类型转换	左
13	*, /, %	乘，除，取余	左
12	+, -	加，减	左
11	<<, >>	移位	左
10	<, >, <=, >=	关系	左
9	==, !=	相等	左
8	&	按位与	左
7	^	按位异或	左
6	|	按位或	左
5	&&	逻辑与	左
4	||	逻辑或	左
3	?:	条件	右
2	=, +=, -=, *=, /=, %=, ^=, <<=, >>=, &=, |=	赋值	右
1	,	顺序计算	左

由于程序员都知道表达式语义的基本数学模型，因此，通常的算法表达式都有良好的合理性。但是不少语言都以不同的方式扩充了操作符集合，优先性常常会被破坏。

2.3.2 语句间的顺序控制

表达式的顺序控制是把操作数和操作符看作基本单位，研究操作符的计算顺序。而语句间的顺序控制是把程序语句作为基本单位，研究语句执行的顺序。程序语言对语句的执行都遵循一条隐含规则，这就是在没有其他顺序控制结构规定的情况下，按照语句在程序中的物理位置执行程序，也就是顺序执行。改变这种语句执行次序的方法是使用程序顺序控制结构，这些控制结构有跳转结构、选择结构和循环结构。

1. 跳转结构

跳转结构就是令程序控制无条件地从当前语句转向给定的语句执行的控制结构，跳转语句的执行非常有效，它反映了计算机本身硬件的转移指令，如 x86 指令中的 jmp 指令。通常的跳转语句都有如下形式：goto <标号>，Fortran 和 C 语言等都提供了 goto 语句。当程序控制遇到 goto 语句时，会转移到标号所指出的相应语句继续执行。

虽然 goto 语句的使用十分简单和高效，但是大量的使用会令程序控制逻辑混乱，程序变得难以理解和维护。人们已经证明可以使用顺序结构、选择结构和循环结构组成任何程序，而抛弃掉“有害的”goto 语句。目前比较一致的观点是，程序员必须谨慎地使用 goto 语句，使用时必须考虑是否可以用更好的结构来代替。

2．选择结构

选择结构是对给定条件进行判断，然后根据结果执行不同的语句或语句块的结构。最典型的选择结构的形式如下：

```
        If <expr> then
            <statements1>
        Else
            <statements2>
        Endif
```

这意味着如果 expr 条件为真，则执行 statements1 语句块，否则执行 statements2 语句块。在某些复杂的情况下，需要对多个条件进行判断，则 if-then-else 语句会进一步复杂，演化为 if-then-elseif-then-else 等。

在两个分支的选择结构基础上，多数语言也会提供多分支的选择结构，它在许多情况下可以改善程序的可读性。典型的多分支选择结构如下：

```
    Switch (<expr>)
    Case of result1:
    <statements1>
    Case of result2:
    <statements2>
    …
    Default:
    <default_statements>
    Endswitch
```

虽然 case 控制结构的功能可以由 if-then-else 结构来模拟，但是 case 控制结构能提供更清晰的计算过程的反映。

C/C++的情况比较特别，在 case 结构中使用 break 语句表示跳出结构的控制，如果在其中一个 case 中没有使用 break 语句，则控制会顺序执行至下一个 case 中的语句。这个特点在为程序员带来方便的同时，也为程序员带来了麻烦。程序员疏忽漏掉的 break 语句会导致程序有意想不到的执行结构。因此，在 C#中不允许这种 case 的“贯穿”，而强制程序员使用 goto 语句跳转至相应的 case 标号，以保证程序员清楚地知道程序控制的行为。

3．循环结构

循环结构是根据条件重复执行指定语句的控制结构。循环结构是由循环头和循环体组成的。循环头就是循环的条件，用于控制循环的次数，循环体则是提供动作的语句。典型的循环头结构有如下几种。

1）计数器循环

这种结构需要说明一个循环计数器，并且在头部说明计数器的初值、终值和增量。典型的计数器循环的结构是 Pascal 的计数器循环：

```
    For I:=0 to 30 step 2 Do <body>
```

该循环的头部说明了计数器为 I，其初值为 0，终值为 30，增量为 2，循环的执行次数为 16 次。

2）条件循环

条件循环是指在给出的条件表达式成立时，重复执行循环体的循环结构，它的头部说明了该条件表达式。这种循环期望在循环体执行时会改变条件测试表达式中的某个变量的值，否则循环将永不终止。典型的条件循环的结构有两种，一种是：

```
While <expr> do <body>
```

另一种是：

```
Repeat <body> until <expr>
```

前者是先测试条件，然后执行循环体，循环体执行零次或零次以上。而后者是先执行循环体，再测试条件，循环体执行一次或以上。

3）基于数据的循环

基于数据的循环的循环次数是由数据格式决定的，例如，C#中的 foreach 结构：

```
foreach (object o in <collection>) {…}
```

对于每一次循环，变量 o 都会取得数据集中的下一个值，数据集元素的个数决定了循环的次数。

4）不定循环

如果循环结束条件过于复杂，不容易在头部表示，通常会使用在循环头部没有显示终止测试的无限循环，然后在循环体中通过条件判断退出循环。C/C++中有两种典型的不定循环结构，一种是：for (;;) <statements>，另一种是：while(1) <statements>。

2.3.3 过程控制

1．过程简介

在程序设计中，习惯把程序看作层次结构，程序从主程序开始执行，然后进入各层次的过程执行，到最后返回主程序结束。过程为程序员提供了一种抽象手段，其实际上是一组输入到一组输出的映射。

过程通常有 4 个要素，分别为过程名、过程体、形式参数列表和返回值类型。例如，C 语言中的函数（即 C 语言中的过程）如下：

```
int Function1 (int x, int y);
```

其中 Function1 为函数名，（int x, int y）为形式参数列表，int 是返回值类型。

2．参数传递方式

当用户调用一个过程时，就会发生通过参数传递信息的过程之间的通信。形式参数就是过程定义中用于命名所传递的数据或其他信息的标识符，而实际参数是在调用点表示向被调用过程传递的数据或其他信息的表达式。在大多数的语言中，形式参数和实际参数之间的对应关系通常按位置来确定。程序语言传递参数的方式通常有传值调用、引用调用。

1）传值调用

在按传值调用时，过程的形式参数取得的是实际参数的值。在这种情况下，形式参数实际上是过程中的局部量，其值的改变不会导致调用点所传送的实际参数的值发生改变，也就是说数据的传送是单向的。在 C 语言中只有按值调用的过程参数传递方式。

2）引用调用

在按引用调用时，过程的形式参数取得的是实际参数所在单元的地址。在过程中，对该形式参数的引用相当于对实际参数所在的存储单元的地址引用。任何改变形式参数值的操作会反映在该存储单元中，也就是反映在实际参数中，因此，数据的传送是双向的。C++语言既支持按值调用，也支持按引用调用。

2.4 程序语言的种类、特点及适用范围

1．按程序设计范型分类

按照程序设计范型的分类，程序设计语言基本上可以分为命令式程序设计语言、函数式程序设计语言、面向对象程序设计语言和逻辑程序设计语言。

命令式程序设计语言是基于动作的语言，计算在这里被看作一个动作的序列。这些动作能够改变变量的值，最典型的动作就是赋值。命令式程序设计语言的代表有：Fortran、Pascal 和 C 语言等。

函数式程序设计语言的代表有 Lisp、ML 等。

面向对象程序设计语言中最核心的内容是对象和类的概念。面向对象的三个核心概念是封装、继承和多态。面向对象程序设计语言的代表有 C++、SmallTalk、Java 等。

逻辑程序设计语言的代表有 Prolog。

2．几种语言的特点及适用范围

1）C++

C++的特点是既支持面向对象程序设计的概念，也支持原来在 C 语言中的过程式程序设计，因此，也有人称其为混合式的面向对象语言。C++支持的面向对象概念包括类、继承、多态、模板、多重继承等。C++在增加了这些面向对象的概念支持后，生成的目标程序与 C 语言生成的相同功能的目标程序的效率相差不超过 10%，是一种极其高效的语言。由于这些特点，以及其在各种计算机系统中被广泛支持，C++语言大量应用于系统程序的设计，包括嵌入式、桌面式和服务器操作系统的设计，大型软件系统的核心模块的设计，以及各类桌面软件的设计。

2）Java

Java 是一个纯面向对象的程序设计语言。Java 与 C++不同，不允许有独立于类存在的过程，所有的概念都必须使用类表达。Java 为了提高代码质量和安全性，去掉了 C++中的指针概念，而完全使用引用的概念。另外，为了提高程序可靠性，Java 提供了内存收集机制，动态内存的管理完全由系统接管。Java 的一个最大的特点是一种半解释语言。编译程序首先把原程序编译为中间代码，然后通过不同平台上的 Java 虚拟机（Java VM）解释执行这些中间代码。较新的方式是不同平台上的 Java 虚拟机把这些中间代码编译为本级代码（Native Code）再执行，以提高执行速度。因此，Java 语言提供了强大的跨平台能力，尤其适用于互联网上的信息系统的开发。

3）Lisp

Lisp 是表处理（List Processing）的截头缩写词，它是函数式程序设计语言。在 Lisp 中，所有的操作均通过表操作进行，变量的赋值也是通过表操作的副作用进行的。Lisp 的初始

设计是为了做符号处理。它被用于各种符号演算：微分和积分演算，电子电路理论，数理逻辑，游戏推演，以及人工智能的其他领域。

4）Prolog

Prolog 程序以特殊的逻辑推理形式回答用户的查询。Prolog 程序具有逻辑的简洁性和表达能力。实际应用上多用于数据库和专家系统。

5）Python

Python 是一种面向对象、解释型计算机程序设计语言，由 Guido van Rossum 于 1989 年底发明，第一个公开发行版发行于 1991 年。Python 语法简洁而清晰，具有丰富和强大的类库。它的昵称是胶水语言。它能够把用其他语言制作的各种模块（尤其是 C/C++）很轻松地联结在一起。常见的一种应用情形是，使用 Python 快速生成程序的原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写，比如 3D 游戏中的图形渲染模块，性能要求特别高，就可以用 C++重写。

6）C#

C#是微软公司发布的一种面向对象的、运行于.NET Framework 之上的高级程序设计语言，并定于在微软职业开发者论坛（PDC）上登台亮相。C#是微软公司研究员 Anders Hejlsberg 的最新成果。C#看起来与 Java 有着惊人的相似；它包括了诸如单一继承、接口、与 Java 几乎同样的语法和编译成中间代码再运行的过程。但是 C#与 Java 有着明显的不同，它借鉴了 Delphi 的一个特点，与 COM（组件对象模型）是直接集成的，而且它是微软公司.NET Windows 网络框架的主角。

第 3 章

操作系统基础知识

操作系统是管理和控制计算机硬件与软件资源的计算机程序。其职能主要包括五个大的方面：处理机管理（进程管理）、存储管理、设备管理、文件管理、作业管理。本章将从这五大方面展开论述。

3.1　操作系统的功能、类型和层次结构

操作系统的定义、功能、类型和层次结构虽然在历年试题中没有涉及，但这是理解操作系统的工作机制的基础，需要深入理解和掌握。重点理解操作系统的定义和功能。

1．操作系统定义

任何一个计算机系统都由两个部分组成：计算机软件系统和计算机硬件系统。操作系统（Operating System，OS）是计算机系统中的核心系统软件，负责管理和控制计算机系统中软件和硬件资源，合理地组织计算机工作流程和有效利用资源，在计算机与用户之间起接口的作用，如图 3-1 所示。

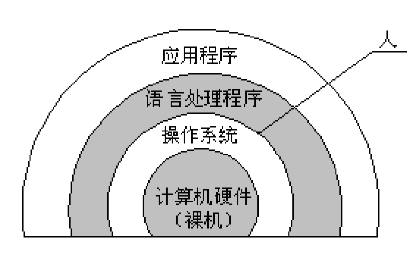


图 3-1　操作系统与软/硬件的关系

在计算机系统中引入操作系统的目的可以从 4 个方面来理解。

1）*用户观点*

操作系统是用户与计算机之间的接口。一方面，用户可以透明地使用计算机软/硬件资源；另一方面，操作系统提供了一些功能强大的系统调用，用户软件可以使用这些系统调用请求操作系统服务。

2）*资源管理观点*

操作系统是计算机资源的管理者，它管理和分配计算机系统软件和硬件资源，合理地组织计算机的工作流程，使资源能为多个用户共享，当用户程序和其他程序争用这些资源时，提供有序的和可控的分配。

3）进程观点

操作系统由一个系统核心和若干并发运行的程序组成。这些运行的程序称为“进程”，进程可以分为用户进程和系统进程两大类。每个进程完成特定的任务，系统核心则控制和协调这些进程的运行。

4）分层观点

操作系统通常采用分层结构实现，各层次的程序按照一定的结构组织并协调工作。

2．操作系统分类

操作系统的基本类型有：

- 批处理操作系统（Batch Processing Operating System）。
- 分时操作系统（Time Share Operating System）。
- 实时操作系统（Real Time Operating System）。
- 网络操作系统（Network Operating System）。
- 分布式操作系统（Distributed Operating System）。

3．操作系统的功能

从资源管理的观点看，操作系统的功能分成五大部分，即处理机管理、存储管理、文件管理、设备管理和作业管理。这五大部分相互配合，协调工作，实现对计算机系统的资源管理和控制程序的执行，为用户提供方便的使用接口和良好的运行环境。

3.2 处理机管理（进程管理）

进程管理是操作系统部分的核心内容，也是历年的考试重点，从 1991 年到 2003 年，共有 10 题涉及进程管理的知识点，占操作系统总题量的 50%。从历年的考查情况看，主要偏重于进程的同步与互斥、信号量和 P－V 操作、进程的基本概念、管程，以及线程等方面。考生对进程部分的知识点应全面掌握。

1．进程的概念

进程是可以与其他程序并发执行的段程序的一次执行过程，是系统进行资源分配和调度的基本单位。进程是一个程序关于某个数据集的一次运行。也就是说，进程是运行中的程序，是程序的一次运行活动。相对于程序，进程是一个动态的概念，而程序是静态的概念，是指令的集合。因此，进程具有动态性和并发性。

从静态的角度看，进程实体由程序块、进程控制块（简称 PCB）和数据块三部分组成。程序块描述该进程所要完成的任务；数据块包括程序在执行时所需要的数据和工作区。进程控制块包括进程的描述信息、控制信息、资源管理信息和 CPU 现场保护信息等，反映了进程的动态特性，如图 3-2 所示。

进程标识	状态	优先级	控制信息	队列	访问权限	现场

图 3-2　进程控制块 PCB

PCB 是进程存在的唯一标志，PCB 描述了进程的基本情况。系统根据 PCB 感知进程的存在和通过 PCB 中所包含的各项变量的变化，掌握进程所处的状态以达到控制进程活动

的目的。在创建一个进程时，首先创建其 PCB，然后才能根据 PCB 中的信息对进程实施有效的管理和控制。当一个进程完成其功能后，系统则释放 PCB，进程也随之消亡。一般情况下，进程的 PCB 结构都是全部或部分常驻内存的。

2．进程的状态转换与控制

1）进程的状态及其转化

- 就绪状态。指进程分配到除处理机以外的必需的资源（已经具备了执行的条件）的状态。进程被创建后处于就绪状态，处于就绪状态的进程可以有多个。
- 执行状态。指进程占有处理机正在CPU上执行的状态。在单CPU系统中，每一时刻只有一个进程处于执行状态。
- 阻塞状态。指进程因等待某个事件的发生而放弃处理机进入等待状态。系统中处于这种状态的进程可以有多个。

现代操作系统还有挂起状态，进程的状态及转换如图 3-3 所示。

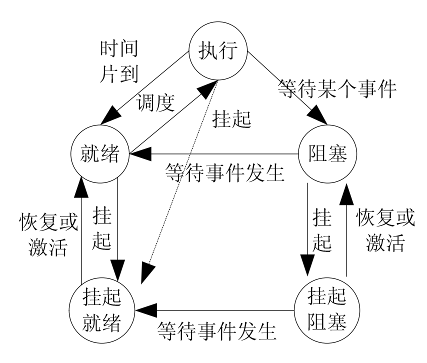


图 3-3　进程的状态及转换

进程的状态随着自身的推进和外界的变化而变化。比如，就绪状态的进程被进程调度程序选中进入执行状态；执行状态的进程因等待某一事件的发生转入等待状态；等待状态的进程在等待事件来到后便进入就绪状态。进程的状态可以动态地相互转换，但阻塞状态的进程不能直接进入执行状态，就绪状态的进程不能直接进入阻塞状态。在任何时刻，任何进程都处于且只能处于某一状态。

2）进程控制

进程控制是通过进程控制原语实现的。用于进程控制的原语主要包括，创建原语、阻塞原语、撤销原语、唤醒原语、优先级原语和调度原语。在操作系统中，原语是一个不可分割的基本单位。它们可以被系统本身调用，有的也以软中断形式供用户进程调用。

创建原语创建一个进程，包括系统创建和父进程创建都必须调用创建原语。新建立的进程开始处于就绪状态。调度原语是按照确定的算法，从就绪队列中选择一个就绪进程，将处理器分配给它；修改这个进程的 PCB 内容。唤醒原语负责叫醒阻塞队列具备运行条件的某进程，使其回到就绪队列。撤销原语将执行完毕的进程登记、回收资源并撤销这个进程及其子进程。

通常操作系统中设置三种队列：执行队列、就绪队列和阻塞队列。在单处理器系统中执行队列只有一个成员。一般阻塞队列的个数取决于等待事件（原因）的个数。新创建的进程处于就绪队列。

3．进程互斥与同步及 P－V 操作

1）进程互斥与同步的定义

进程互斥定义为，一组并发进程中一个或多个程序段，因共享某一公有资源而导致它们必须以一个不允许交叉执行的单位执行。也就是说，互斥是要保证临界资源在某一时刻只被一个进程访问。

进程同步定义为，异步环境下的一组并发进程因直接制约而互相发送消息，进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程同步。也就是说，进程之间是异步执行的，同步即是使各进程按一定的制约顺序和速度执行。

2）信号量（Semaphore）与 P－V 操作

信号量可以有效地实现进程的同步和互斥。在操作系统中，信号量是一个整数。当信号量大于等于零时，代表可供并发进程使用的资源实体数，当信号量小于零时则表示正在等待使用临界区的进程数。建立一个信号量必须说明所建信号量所代表的意义和设置初值，以及建立相应的数据结构，以便指向那些等待使用该临界区的进程。

对信号量只能施加特殊的操作：P 操作和 V 操作，P 操作和 V 操作都是不可分割的原子操作，也称为原语，因此，P－V 原语执行期间不允许中断发生。

P（sem）操作的作用是将信号量 sem 值减 1，若 sem 的值成负数，则调用 P 操作的进程暂停执行，直到另一个进程对同一信号量做 V 操作。V（sem）操作的作用是将信号量 sem 值加 1，若 sem 的值小于等于 0，从相应队列（与 sem 有关的队列）中选择一个进程，唤醒它。

一般 P 操作与 V 操作的定义如下。

P 操作：

```
P(sem){
sem = sem－1;
if(sem<0) 进程进入等待状态;
else  继续进行; }
```

V 操作：

```
V(sem){
sem = sem＋1;
if(sem≤0) 唤醒队列中的一个等待进程;
else  继续进行; }
```

3）用 P－V 操作实现进程互斥

为了保护共享资源（如公共变量等），使它们不被多个进程同时访问，就要阻止这些进程同时执行访问这些资源的代码段，这些代码段称为临界区，这些资源称为临界资源；进程互斥不允许两个以上共享临界资源的并发进程同时进入临界区。利用 P－V 原语和信号量可以方便地解决并发进程对临界区的进程互斥问题。

设信号量 mutex 是用于互斥的信号量，初值为 1，表示没有并发进程使用该临界区。于是各并发进程的临界区可改写成如下形式的代码段：

```
   P(mutex);
临界区
   V(mutex);
```

4）用 P－V 操作实现进程同步

要用 P－V 操作实现进程同步，需要引进私用信号量。私用信号量只与制约进程和被制约进程有关，而不是与整组并发进程相关。与此相对，进程互斥使用的信号量为公用信号量。首先为各并发进程设置私用信号量，然后为私用信号量赋初值，最后利用 P－V 原语和私用信号量规定各进程的执行顺序。

经典同步问题的例子是生产者－消费者问题。这要求存后再取，取后再存，即有两个制约关系。为此，需要两个信号量，记为 Bufempty 和 Buffull，它们的初值分别是 1 和 0，相应的程序段形式如下：

• 生产者

```
loop
生产一产品 next;
P(Bufempty);
next 产品存缓冲区;
V(Buffull);
endloop
```

• 消费者

```
loop
P(Buffulll);
从缓冲区中取产品;
V(Bufempty);
使用产品
endloop
```

4．进程通信与管程

1）进程通信

通信（Communication）就是在进程间传送数据。一般来说，进程间的通信根据通信内容可以划分为两种：控制信息的传送和大批量数据的传送。把控制信息的传送称为低级通信，而把大批量数据的传送称为高级通信。进程的同步和互斥是通过信号量进行通信来实现的，属于低级通信。高级通信原语则提供两种通信方式：有缓冲区的通信和无缓冲区的通信。

2）管程

汉森（Brinch Hansen）和霍尔（Hoare）提出了一个新的同步机制——管程。管程是一个由过程、变量及数据结构等组成的集合，即把系统中的资源用数据抽象地表示出来。这样，对资源的管理就可以用数据及在其上实施操作的若干过程来表示，而代表共享资源的数据及在其上操作的一组过程就构成了管程。进程可以在任何需要资源的时候调用管程，且在任一时刻最多只有一个进程能够真正地进入管程，而其他调用进程则只能等待。由此看来，管程实现了进程之间的互斥，使临界区互斥实现了自动化，它比信号量更容易保证并发进程的正确性。管程结构如图 3-4 所示。

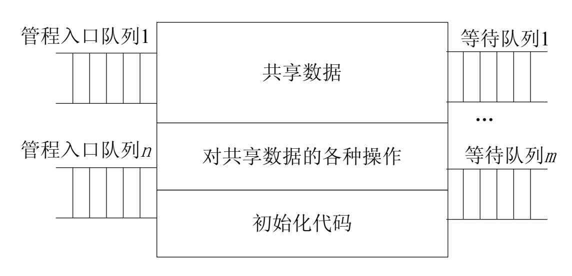


图 3-4　管程结构

5．进程调度与死锁

1）进程调度

进程调度即处理器调度（又称为上下文转换），它由调度原语实现。进程调度的方式有两类：剥夺方式与非剥夺方式。所谓非剥夺方式是指一旦某个作业或进程占有了处理器，别的进程就不能把处理器从这个进程手中夺走，直到该进程自己因调用原语操作而进入阻塞状态，或时间片用完而让出处理机。剥夺方式即就绪队列中一旦有进程优先级高于当前执行进程优先级时，便立即发生进程调度，转让处理机。

2）进程调度算法

进程调度的算法是服务于系统目标的策略，对于不同的系统与系统目标，常采用不同的调度算法，如：

- 先来先服务（FCFS）调度算法，又称为先进先出（FIFO）。就绪队列按先来后到的原则排队。
- 优先数调度。优先数反映了进程优先级，就绪队列按优先数排队，有两种确定优先级的方法，即静态优先级和动态优先级。静态优先级是指进程的优先级在进程开始执行前确定，执行过程中不变，而动态优先级则可以在进程执行过程中改变。
- 轮转法（Round Robin）。就绪队列按FCFS方式排队。每个进程执行一次占有处理器时间都不超过规定的时间单位（时间片）。若超过，则自行释放自己所占有的CPU而排到就绪队列的末尾，等待下一次调度。同时，进程调度程序又去调度当前就绪队列中的第一个进程。

3）死锁

当若干个进程互相竞争对方已占有的资源，无限期地等待，不能向前推进时，会造成“死锁”。死锁是系统的一种出错状态，应该尽量预防和避免。产生死锁的主要原因是供共享的系统资源不足、资源分配策略和进程的推进顺序不当。

产生死锁的必要条件是：互斥条件，保持和等待条件，不剥夺条件，以及环路等待条件。

解决死锁有两种策略：一种是在死锁发生前采用的预防和避免策略；另一种是在死锁发生后采用的检测与恢复策略。

死锁的预防主要是通过打破死锁产生的四个必要条件之一来保证不会产生死锁。死锁避免策略，则是在系统进行资源分配时，先执行一个死锁避免算法（典型的如银行家算法），来保证本次分配不会导致死锁的发生。实际上系统出现死锁的概率很小，故从系统所花的代价上看，采用死锁发生后的检测与恢复策略要比采用死锁发生前的预防与避免策略代价小一些。

6．线程

在支持线程的操作系统中，线程是进程中的一个实体，是系统实施调度的独立单位。线程只拥有一些在运行中必不可少的资源，它与属于同一个进程的其他线程共享该进程所拥有的资源。各线程之间可以并发地运行。线程切换时只需保存和设置少量寄存器的内容，而并不涉及存储器管理方面的操作，所以线程切换的开销远远小于进程的切换（原运行进程状态的切换还要引起资源转移及现场保护等问题）。同一个进程中的多个线程共享同一个地址空间，这使得线程之间同步和通信的实现也比较容易。

3.3 存储管理

存储管理的主要对象是内存，是除处理器外操作系统管理的最重要的资源，也是历年考核的重点。存储管理部分偏重于虚拟存储、分区存储、地址转换及交换技术等知识。

1．存储管理的概念

存储管理主要是指对内存储器的管理，负责对内存的分配和回收、内存的保护和内存的扩充。存储管理的目的是尽量提高内存的使用效率。

2．单一连续区管理

在单道程序系统中，内存区域的用户空间全部为一个作业或进程占用。单一连续分配方法主要用于早期单道批处理系统。单一连续分配方法主要采用静态分配方法，为降低成本和减少复杂度，通常不对内存进行保护，因而会引起冲突使系统瘫痪。

3．分区存储管理

分区存储管理包括固定分区、可变分区，其基本思想是把内存划分成若干个连续区域，每个分区装入一个作业运行。要求作业一次性装入内存，且分区内部地址必须连续。

1）固定分区存储管理

固定分区分配方法是把内存空间固定地划分为若干个大小不等的区域，划分的原则由系统决定。系统使用分区表描述分区情况。分区一旦划分结束，在整个执行过程中每个分区的长度和内存的总分区个数保持不变。

2）可变分区存储管理

可变分区分配方法是把内存空间按用户要求动态地划分成若干个分区。随着进程的执行，剩余的自由区域会变得更小，这时需要合并自由区和存储拼接技术。合并自由区是将相邻自由存储区合并为单一自由区的方法；存储拼接技术也称为碎片收集，包括移动存储器的所有被占用区域到主存的某一端。可变分区克服了固定分区分配方法中的小作业占据大分区后产生碎片的浪费问题。

3）存储分配算法

常使用的 4 种存储分配算法介绍如下。

- 首次适应算法：把内存中的可用分区单独组成可用分区表或可用分区自由链，按起始地址递增的次序排列。每次按递增次序向后找，一旦找到大于或等于所要求内存长度的分区，则结束探索，从找到的分区中找出所要求的内存长度分配给用户，并把剩余的部分进行合并。
- 循环适应算法：上述首次适应法经常利用的是低地址空间，后面经常是较大的空白

区，为使内存所有线性地址空间尽可能轮流使用到，每重新分配一次，都在当前之后寻找。

- 最佳适应算法：最佳适应算法是将输入作业放入主存中与它所需大小最接近的空白区中，使剩下的未用空间最小。该法要求空白区大小按从小到大次序组成空白区可用表或自由链。在进行分配时总是从最小的一个开始查询，因而找到的一个能满足要求的空白区便是最佳的一个。
- 最差适应算法：分配时把一个作业程序放入主存中最不适合它的空白区，即最大的空白区（空闲区）内。

4）交换与覆盖技术

覆盖与交换技术是在多道程序环境下用来扩充内存的两种方法。覆盖技术主要用在早期的操作系统中，而交换技术则在现代操作系统中得到了进一步发展。

覆盖技术是一种解决小内存运行大作业的方法。一个作业中若干程序段和数据段可以不同时使用，这样它们就可以共享内存的某个区域，再根据需要分别调入该区域，这个区域就称为覆盖区。将程序执行时并不要求同时装入主存的覆盖组成一组，并称其为覆盖段，这个覆盖段分配到同一个覆盖区。

交换技术可以将暂不需要的作业移到外存，让出内存空间以调入其他作业，交换到外存的作业也可以被再次调入。交换技术与覆盖技术相比不要求给出程序段之间的覆盖结构。交换主要是在作业之间进行的，而覆盖则主要是在同一个作业内进行的。

4．页式存储管理

分页的基本思想是把程序的逻辑空间和内存的物理空间按照同样的大小划分成若干页面，以页面为单位进行分配。在页式存储管理中，系统中虚地址是一个有序对（页号、位移）。系统为每一个进程建立一个页表，其内容包括进程的逻辑页号与物理页号的对应关系、状态等。

5．段式存储管理

段式存储管理与页式存储管理相似。分段的基本思想是把用户作业按逻辑意义上有完整意义的段来划分，以段为单位作为内、外存交换的空间尺度。一个作业是由若干个具有逻辑意义的段（如主程序、子程序、数据段等）组成的。在分段系统中，容许程序（作业）占据内存中许多分离的分区。每个分区存储一个程序分段。这样，每个作业需要几对界限地址寄存器，判定访问地址是否越界也变得困难。在分段存储系统中常常利用存储保护键实现存储保护。分段系统中虚地址是一个有序对（段号、位移）。系统为每个作业建立一个段表，其内容包括段号、段长、内存起始地址和状态等。状态指出这个段是否已调入内存，即内存起始地址指出这个段，状态指出这个段的访问权限。

6．段页式存储管理

段页式管理是段式和页式两种管理方法结合的产物，综合了段式组织与页式组织的特点，根据程序模块分段，段内再分页，内存被划分成定长的页。段页式系统中虚地址形式是（段号、页号、位移）。系统为每个进程建立一个段，为每个段建立一个页表。段页式管理采用段式分配、页式使用的方法，便于动态连接和存储的动态分配。这种存储管理能提高内存空间的利用率。段页式虚拟存储管理结合了段式和页式的优点，但增加了设置表格（段表、页表）和查表等开销，段页式虚拟存储器一般只在大型计算机系统中使用。

7．页面调度

如果选择的页面被频繁地装入和调出，这种现象称为"抖动"，应减少和避免抖动现象。常用的页面调度算法有如下几种。

- 最优（OPT）算法。选择不再使用或最远的将来才被使用的页，难以实现，常用于淘汰算法的比较。
- 随机（RAND）算法。随机地选择被淘汰的页，开销小，但是可以选中立即就要访问的页。
- 先进先出（First In First Out，FIFO）算法，又称为轮转法（RR）。选择在内存驻留时间最长的页，似乎合理，但可能淘汰掉频繁使用的页。另外，使用FIFO算法时，在未给予进程分配足够的页面数时，有时会出现给予进程的页面数增多，缺页次数反而增加的异常现象。FIFO算法简单，可采用队列实现。
- 最近最少使用（Least Recently Used，LRU）算法。选择离当前时间最近的一段时间内使用的最少的页。这个算法的主要出发点是，如果某个页被访问了，则它可能马上就要被访问；反之，如果某个页长时间未被访问，则它在最近一段时间也不会被访问。

另外，还有最不经常使用的页面先淘汰（Least Frequent Used，LFU）、最近没有使用的页面先淘汰（NUR）、最优淘汰算法（OPT）等。

3.4 设备管理

设备管理是指操作系统对除 CPU 和内存之外所有设备的管理。历年考题中主要涉及缓冲技术和 SPOOLing 技术等知识点。

1．设备管理的概念

在计算机系统中，除处理器和内存之外，其他的大部分硬件设备称为外部设备。它包括输入/输出设备、辅存设备及终端设备等。为了完成上述主要任务，设备管理程序一般要提供下述功能。

- 提供和进程管理系统的接口。当进程要求设备资源时，该接口将进程要求转达给设备管理程序。
- 进行设备分配。按照设备类型和相应的分配算法把设备和其他有关的硬件分配给请求该设备的进程，并把未分配到所请求设备或其他有关硬件的进程放入等待队列。
- 实现设备和设备、设备和CPU等之间的并行操作。
- 进行缓冲区管理。主要减少外部设备和内存与CPU之间的数据速度不匹配的问题，系统中一般设有缓冲区（器）来暂放数据。设备管理程序负责进行缓冲区分配、释放及有关的管理工作。

2．数据传输控制方式

外围设备和内存之间的常用数据传送控制方式介绍如下。

- 程序控制方式。
- 中断方式。
- 直接存储访问（DMA）方式。指外部设备和内存之间开辟直接的数据交换通路。
- 通道方式。通道又称为输入/输出处理器（IOP），主要有三类通道：字节多路通道、选择通道和成组多路通道。

3．设备的分配

1）设备分配原则

设备分配方式有两种：一种是静态分配；另一种是动态分配。静态分配方式是在用户作业开始执行之前，由系统一次分配该作业所要求的全部设备、控制器和通道。一旦分配之后，这些设备、控制器和通道就一直为该作业所占用，直到该作业被撤销。静态分配方式不会出现死锁，但是设备的使用效率低。

动态分配在进程执行过程中根据执行需要来进行。当进程需要设备时，通过系统调用命令向系统提出设备请求，由系统按照事先规定的策略给进程分配所需要的设备、I/O 控制器和通道，一旦用完之后，便立即释放。动态分配方式有利于提高设备的利用率，但如果分配算法使用不当，则有可能造成进程死锁。

2）设备分配策略

常用的分配策略有先请求先分配、优先级高者先分配策略等。

优先级高者先分配策略和进程调度的优先级算法是一致的，即进程的优先级高，那么它的 I/O 请求也优先满足。对于相同优先级的进程来说，则按照先请求先分配策略分配。

4．磁盘调度算法

访问磁盘的时间因子由 3 部分构成，它们是查找（查找磁道）时间、等待（旋转等待扇区）时间和数据传输时间，其中查找时间是决定因素。因此，磁盘调度算法主要有如下几种。

- 先来先服务（FCFS）调度。按先来后到次序服务，未做优化。
- 最短查找时间优先（SSTF）调度。SSTF查找距离磁头最短（也就是查找时间最短）的请求作为下一次服务的对象。SSTF查找模式有高度局部化的倾向，会推迟一些请求的服务，甚至引起无限拖延（又称为饥饿）。
- SCAN调度。又称为电梯算法，SCAN算法是磁头前进方向L的最短查找时间优先算法，它排除了磁头在盘面局部位置上的往复移动，SCAN算法在很大程度上消除了SSTF算法的不公平性，但仍有利于对中间磁道的请求。

5．虚设备与 SPOOLing 技术

SPOOLing（Simultaneous Peripheral Operation On Line）的意思是外部设备同时联机操作，又称为假脱机输入/输出操作，采用一组程序或进程模拟一台输入/输出处理器。SPOOLing 系统的组成如图 3-5 所示。该技术利用了专门的外围控制机将低速 I/O 设备上的数据传送到高速设备上，或者相反。但是当引入多道程序后，完全可以利用其中的一道程序来模拟脱机输入时的外围控制机的功能，把低速的I/O设备上的数据传送到高速磁盘上；再利用另一道程序来模拟脱机输出时的外围控制机的功能，把高速磁盘上的数据传送到低速的 I/O 设备上。这样便可以在主机的控制下实现脱机输入、输出的功能。此时的外围操作与 CPU 对数据的处理同时进行。将这种在联机情况下实现的同时外围操作称为 SPOOLing，或称为假脱机操作。

采用假脱机技术，可以将低速的独占设备改造成一种可共享的设备，而且一台物理设备可以对应若干台虚拟的同类设备。SPOOLing 系统必须有高速、大容量并且可随机存取的外存（如磁盘或磁鼓）支持。

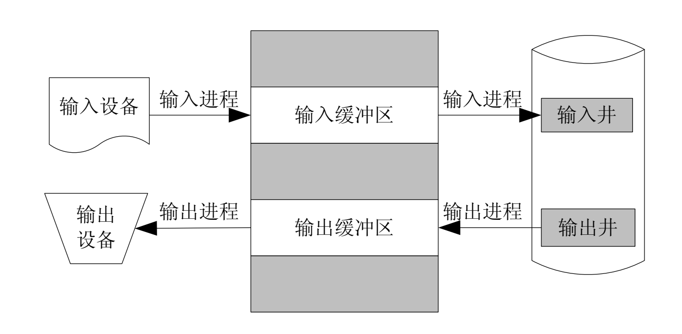


图 3-5　SPOOLing 系统的组成

3.5　文件管理

与存储管理相对，文件管理是对外部存储设备上的以文件方式存放的信息的管理。核心内容是文件的结构和访问方式、存储空间管理及目录结构等知识点。从历年试题看，有涉及存储空间管理和与 UNIX 具体的文件系统相关的试题。

1．文件和文件系统的概念

1）文件和文件系统

文件是信息的一种组织形式，是存储在辅助存储器上的具有标识名的一组信息集合。它可以是有结构的，也可以是无结构的。操作系统中由文件系统来管理文件的存储、检索、更新、共享和保护。文件系统包括两个方面，一方面包括负责管理文件的一组系统软件，另一方面包括被管理的对象——文件。

2）文件类型

根据不同的方面，文件有多种分类方法：

- 按文件的用途可以分为系统文件、库文件和用户文件等。
- 按文件的安全属性可分为只读文件、读/写文件、可执行文件和不保护文件等。
- 按文件的信息流向可以分为输入文件、输出文件和输入/输出文件等。
- 按文件的组织形式可以分为普通文件、目录文件和特殊文件等。特殊文件是UNIX系统采用的技术，把所有的输入/输出设备都视为文件（特殊文件）。特殊文件的使用形式与普通文件相同。

2．文件的结构和存取方式

1）文件的结构

文件的结构是指文件的组织形式，从用户观点所看到的文件组织形式，称为文件的逻辑结构。一般文件的逻辑结构可以分为两种，无结构的字符流文件和有结构的记录文件。记录文件由记录组成，即文件内的信息划分成多个记录，以记录为单位组织和使用信息。记录文件有顺序文件、索引顺序文件、索引文件和直接文件。

文件的物理结构是指文件在存储设备上的存放方法。文件的物理结构侧重于提高存储器的利用效率和降低存取时间。文件的存储设备通常划分为大小相同的物理块，物理块是

分配和传输信息的基本单位。文件的物理结构涉及文件存储设备的组织策略和文件分配策略，决定文件信息在存储设备上的存储位置。常用的文件分配策略有顺序分配（连续分配）、链接分配（串联分配）和索引分配。

2）文件的访问方式

用户通过对文件的访问（读写）来完成对文件的查找、修改、删除和添加等操作。常用的访问方法有两种，即顺序访问和随机访问。

3．文件存储设备管理

文件存储设备管理，就是操作系统要有效地进行存储空间的管理。由于文件存储设备是分成许多大小相同的物理块，并以块为单位交换信息，因此，文件存储设备的管理实质上是对空闲块的组织和管理问题，它包括空闲块的组织、空闲块的分配与空闲块的回收等问题。有三种不同的空闲块管理方法，它们是索引法、链接法和位图法。

4．文件控制块和文件目录

1）文件控制块

文件控制块是系统在管理文件时所必需的信息的数据结构，是文件存在的唯一标志，简称为 FCB。文件目录就是文件控制块的有序集合。FCB 的内容包括相应文件的基本属性，大致可以分成 4 个部分。

- 基本信息：如文件名、文件类型和文件组织等。
- 保护信息：如口令、所有者名、保存期限和访问权限等。
- 位置信息；如存储位置、文件长度等。
- 使用信息：如时间信息、最迟使用者等。

2）文件目录

文件控制块的集合称为文件目录，文件目录也被组织成文件，常称为目录文件。

文件管理的一个重要方面是对文件目录进行组织和管理。文件系统一般采用一级目录结构、二级目录结构和多级目录结构。DOS、UNIX、Windows 系统都是采用多级（树型）目录结构。

5．文件的操作与使用

1）文件的使用

一般文件系统提供一组专门用于文件、目录的管理命令，如目录管理、文件控制和文件存取等命令。

- 目录管理命令：如建立目录、显示工作目录、改变目录、删除目录等。
- 文件控制命令：如建立文件、删除文件、打开文件、关闭文件、改文件名、改变文件属性等。
- 文件存取命令：如读/写文件、显示文件内容、复制文件等。

2）文件共享和安全

文件的共享是指不同的用户使用同一文件。文件的安全是指文件的保密和保护，即限制非法用户使用和破坏文件。

文件的共享可以采用文件的绝对路径名或相对路径名共享同一文件。一般的文件系统

要求用户先打开文件，再对文件进行读/写，不再使用时关闭文件。若两个用户可以同时打开文件，对文件进行存取，则称为动态文件共享。

文件的安全管理措施常常在系统级、用户级、目录级和文件级上实施。

6．存储空间管理

一个大容量的文件存储器为系统本身和许多用户所共享。为方便用户“按名存取”所需文件，系统应能自动为用户分配并管理系统和用户的存储空间。为此，必须解决如下三个问题：登记空闲区的分布情况、按需要给一个文件分配存储空间，以及收回不再保留的文件所占的存储空间。上述问题都可以归结为磁盘空闲区的管理问题，常用的磁盘空闲区的管理方法有空闲文件目录、空闲块链、位示图法和成组链接法。

1）空闲文件目录

磁盘空间上一个连续的未分配区域成为空闲文件。系统为所有这些空闲文件单独建立一个目录。对每个空闲文件，在这个目录中建立一个表目。表目的内容包括第一个空闲块地址（物理块号）和空闲块个数等。在进行存储空间的分配时，也可采用首次适应和最佳适应等算法，而回收时，同样要进行空闲区的合并。这种方法的优点是空闲区的分配和回收都相当容易，但用来管理空闲区的空闲表需要占用大量的存储空间。

2）空闲块链

空闲块链是将所有空闲块用链接指针或索引结构组成一个空闲文件。释放和分配空闲块都可以在链首进行，只需要修改几个有关的链接字。该方法只要求在内存中保存一个指针，令它指向第一个空闲块，其优点是实现简单，但工作效率低，因为每当在链上增加或移去空闲块时，都需要对空闲块做较大的调整，从而会有较大的系统开销。一种改进方法是将空闲块分成若干组，再用指针将组与组链接起来，将这种管理空闲块的方法称为成组链接法，它在进行空闲块的分配与回收时要比空闲块链法节省时间。

3）位示图法

位示图是利用二进制的1位来表示文件存储空间中的1个块的使用情况。一个 m 行、n 列的位示图，可用来描述 $m \times n$ 块的文件存储空间，当行号、列号和块号都是从 0 开始编号时，第 i 行、第 j 列的二进制位对应的物理块号为 $i \times n+j$。如果“0”表示对应块空闲，“1”表示对应块已分配，则在进行存储空间的分配时，可顺序扫描位示图，从中找出一个或一组值为“0”的二进制位，将对应的块分配出去，并将这些位置“1”，而在回收某个块时，只需找到对应的位，并将其值清零即可。位示图法适合于所有的分配方式，它简单易行，而且位示图通常较小，故可将其读入内存，从而进一步加快文件存储空间分配和回收的速度。

4）成组链接法

成组链接法是对空闲块链法的一种改进，它将一个文件卷的所有空闲盘块按固定大小（例如，每组 m 块）分成若干组，并将每一组的盘块数和该组所有的盘块记入前一组的最后一个盘块中，第一组的盘块数和该组的所有盘块号则记入超级块的空闲盘块中。当系统要为用户分配文件所需的盘块时，若第一组不只一块，则将超级块中的空闲盘块数减 1，并将空闲盘块栈顶的盘块分配出去；若第一组只剩一块且栈顶的盘块号不是结束标记“0”，则先将该块的内容（记录有下一组的盘块数和盘块号）读到超级块中，然后再将该块分配出去；否则，若栈顶的盘块号为结束标记“0”，则表示该磁盘上已无空闲盘块可供分配。

在系统回收空闲盘块时，若第一组不满 m 块，则只需将回收块的块号填入超级块的空闲盘栈顶，并将其中的空闲盘块数加 1；若第一组已有 m 块，则必须先将超级块中的空闲盘块数和空闲盘块号写入回收块中，然后将盘块数和回收块的块号记入超级块中。

值得注意的是，超级块中的空闲盘块栈是临界资源，对该栈的操作必须互斥地进行。系统需要为空闲盘块设置一把“锁”，并通过上锁和解锁来实现对空闲盘块栈的操作。成组链接法除第一组空闲盘块外，其余空闲盘块的登记不占额外的存储空间，而超级块（即文件卷的第一块）已在安装磁盘时拷入内存，因此，绝大部分的分配和回收工作可在内存中进行，从而使之具有较高的效率。

3.6 作业管理

作业管理包含进程管理，因而对作业管理的理解可以加强对进程管理的理解。历年试题中常将作业调度与进程调度、设备管理综合起来考查，所以应注意对操作系统整体的工作机制的理解和掌握。

1．作业状态与作业管理

操作系统中用来控制作业的进入、执行和撤销的一组程序称为作业管理程序，这些控制功能也能通过把作业细化，通过进程的执行来实现。

1）作业状态及其转换

一个作业从交给计算机系统到执行结束退出系统，一般都要经历提交、后备、执行和完成 4 个状态。其状态转换如图 3-6 所示。

- 提交状态。作业由输入设备进入外存储器（也称为输入井）的过程称为提交状态。处于提交状态的作业，其信息正在进入系统。
- 后备状态。当作业的全部信息进入外存后，系统就为该作业建立一个作业控制块（JCB）。
- 执行状态。一个后备作业被作业调度程序选中分配了必要的资源并进入内存，作业调度程序同时为其建立了相应的进程后，该作业就由后备状态变成执行状态。

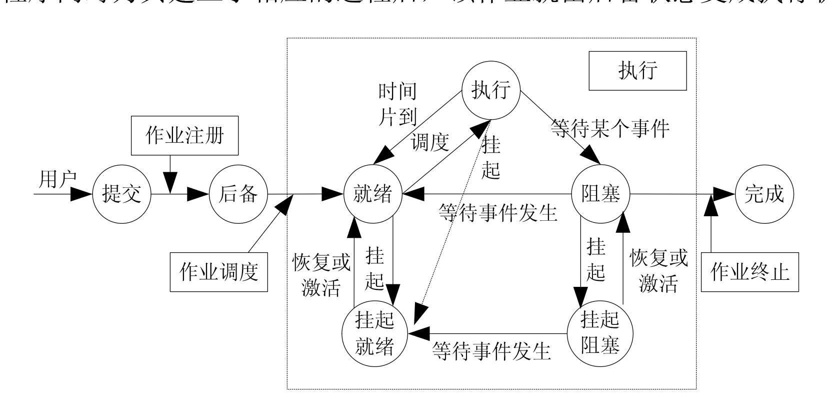


图 3-6　作业的状态及其转换

- 完成状态：当作业正常运行结束，它所占用的资源尚未全部被系统回收时的状态为完成状态。

2）处理机调度

处理机调度通常分为三级调度，即低级调度、中级调度和高级调度。

- 高级调度：高级调度也称为作业调度。高级调度的主要功能是在批处理作业的后备作业队列中选择一个或者一组作业，为它们建立进程，分配必要的资源，使它们能够运行起来。
- 中级调度：中级调度也称为交换调度，中级调度决定进程在内、外存之间的调入、调出。其主要功能是在内存资源不足时将某些处于等待状态或就绪状态的进程调出内存，腾出空间后，再将外存上的就绪进程调入内存。
- 低级调度：低级调度也称为进程调度，低级调度的主要功能是确定处理器在就绪进程间的分配。

3）作业控制块（JCB）

在作业管理中，系统为每一个作业建立一个作业控制块 JCB。系统通过 JCB 感知作业的存在。JCB 包括的主要内容有，作业名、作业状态、资源要求、作业控制方式、作业类型，以及作业优先权等。

2．作业调度及其常用调度算法

作业调度主要完成从后备状态到执行状态的转变，以及从执行状态到完成状态的转变。作业调度算法有如下几种。

- 先来先服务（FCFS）。按作业到达的先后次序调度，它不利于短作业。
- 短作业优先（SJF）。按作业的估计运行时间调度，估计运行时间短的作业优先调度。它不利于长作业，可能会使一个估计运行时间长的作业迟迟得不到服务。
- 响应比高者优先（HRN）。综合上述两者，既考虑作业估计运行时间，又考虑作业等待时间，响应比：HRN=（估计运行时间+等待时间）/估计运行时间。
- 优先级调度。根据作业的优先级别，优先级高者先调度。

3．用户接口

用户接口也称为用户界面，其含义有两种：一种是指用户与操作系统交互的途径和通道，即操作系统的接口；另一种是指这种交互环境的控制方式，即操作环境。

1）操作系统的接口

操作系统的接口又可分成命令接口和程序接口。

- 命令接口：命令接口包含键盘命令和作业控制命令。
- 程序接口：程序接口又称为编程接口或系统调用，程序经编程接口请求系统服务，即通过系统调用，程序与操作系统通信。

系统调用是操作系统提供给编程人员的唯一接口。系统调用对用户屏蔽了操作系统的具体动作而只提供有关功能。系统调用大致分为设备管理、文件管理、进程控制、进程通信和存储管理等。

2）操作环境

操作环境支持命令接口和程序接口，提供友好的、易用的操作平台。操作系统的交互界面已经从早期的命令驱动方式发展到菜单驱动方式、图符驱动方式和视窗操作环境。

3.7 嵌入式操作系统

嵌入式系统通常是指内部包含智能控制器的设备，它具有集成度高、体积小、反应速度快、智能化、稳定及可靠性强等特点。嵌入式实时控制系统必须要非常仔细地研究实时性的保证实施。

嵌入式系统应具有的特点是高可靠性；在恶劣的环境或突然断电的情况下，系统仍然能够正常工作；许多嵌入式应用要求实时性，这就要求嵌入式操作系统具有实时处理能力；嵌入式系统和具体应用有机地结合在一起，它的升级换代也是和具体产品同步进行的；嵌入式系统中的软件代码要求高质量、高可靠性，一般都固化在只读存储器或闪存中，也就是说软件要求固态化存储，而不是存储在磁盘等载体中。

嵌入式软件（Embedded Software），从广义上讲是计算机软件的一种，它也是由程序及其文档组成的。嵌入式软件是嵌入在设备内部并控制设备行为的一种专用软件，其最基本的特点是软件固态化存储在存储芯片或单片机中，而不是存储于磁盘等载体中。嵌入式软件一般在设备启动时自动运行，无须人工干预。通常要求具有实时响应能力，一般不要求复杂的用户界面，也无须用户进行二次开发。

嵌入式软件可分成系统软件、支撑软件（中间件）和应用软件三类，最低层即系统软件，包括操作系统及数据库管理系统。下面定义的嵌入式操作系统、嵌入式数据库、嵌入式中间件和嵌入式应用软件，必须同时符合上述嵌入式软件的定义。

嵌入式操作系统（Embedded Operating System，EOS）是以应用为中心，以计算机技术为基础，软/硬件可裁减，对功能、可靠性、成本、体积、功耗有严格要求的专用性计算机系统。嵌入式操作系统是设备信息系统的核心，管理、监控和维护设备硬件和软件资源，支持和调度各种应用软件的运行，实现处理机管理、内存管理、I/O 设备管理、文件管理及作业管理。

目前，已推出一些应用比较成功的 EOS 产品系列。随着 Internet 技术的发展、信息家电的普及应用及 EOS 的微型化和专业化，EOS 开始从单一的弱功能向高专业化的强功能方向发展。嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固化及应用的专用性等方面具有较为突出的特点。EOS 是相对于一般操作系统而言的，它除具备一般操作系统最基本的功能外，如任务调度、同步机制、中断处理、文件处理等，还有如下特点。

- 可装卸性。开放性、可伸缩性的体系结构。
- 强实时性。EOS实时性一般较强，可用于各种设备控制当中。
- 统一的接口。提供各种设备驱动接口。
- 操作方便、简单，提供友好的图形界面，追求易学易用。
- 提供强大的网络功能，支持TCP/IP协议及其他协议，提供TCP/UDP/IP/PPP协议支持及统一的MAC访问层接口，为各种移动计算设备预留接口。
- 强稳定性，弱交互性。嵌入式系统一旦开始运行就不需要用户过多地干预，这就要求负责系统管理的EOS具有较强的稳定性。嵌入式操作系统的用户接口一般不提供操作命令，它通过系统的调用命令向用户程序提供服务。
- 固化代码。在嵌入式系统中，嵌入式操作系统和应用软件被固化在嵌入式系统计算机的ROM中。辅助存储器在嵌入式系统中很少使用，因此，嵌入式操作系统的文件管理功能应该能够很容易地拆卸，而用各种内存文件系统。

- 更好的硬件适应性，也就是良好的移植性。

国际上用于信息电器的嵌入式操作系统有 40 种左右。现在，市场上非常流行的 EOS 产品，包括 3Com 公司下属子公司的 Palm OS，Microsoft 公司的 Windows CE 和开放源代码的 Linux。

第 4 章

软件工程基础知识

根据考试大纲，要求考生掌握软件生命周期各阶段的任务，结构化分析和设计方法、面向对象的分析与设计，软件开发工具与环境的基础知识，软件质量保证的基础知识，软件过程改进与评估，以及软件项目管理基础知识七个方面的知识。

4.1　软件生命周期与软件开发模型

本节将介绍软件生命周期与软件开发模型。

4.1.1　软件危机与软件工程

软件工程是一门年轻的学科。“软件工程”这个概念最早是在 1968 年召开的一个当时被称为“软件危机”的会议上提出的。自 1968 年以来，我们在该领域已经取得了长足的进步。软件工程的发展已经极大地完善了我们的软件，使我们对软件的开发活动也有了更深的理解。

1．软件危机

那么什么是软件危机呢？自从强大的第三代计算机硬件问世以后，许多原来难以实现的计算机应用成为现实，同时对软件系统的需求数量和复杂度要求变得更高。而当时的软件开发技术无法满足这一日益增长的需求，引发了软件危机。它主要表现为：

- 软件开发生产率提高的速度远远跟不上计算机迅速普及的趋势。软件需求的增长得不到满足，软件产品“供不应求”的现象使人类无法充分利用现代计算机硬件提供的巨大潜力。
- 软件成本在计算机系统总成本中所占的比例逐年上升。
- 不能正确估计软件开发产品的成本和进度，致使实际开发成本高出预算很多，而且超出预期的开发时间要求。
- 软件开发人员和用户之间的信息交流往往很不充分，用户对“已完成的”软件系统不满意的现象经常发生。
- 软件产品的质量不易保证。
- 软件产品常常是不可维护的。
- 软件产品的重用性差，同样的软件多次重复开发。
- 软件通常没有适当的文档资料。

软件危机产生的原因一方面是软件开发本身的复杂性，另一方面是其与当时的手工作坊式软件开发模式有密切关系。

2．软件工程

开发一个具有一定规模和复杂性的软件系统和编写一个简单的程序大不一样。其间的差别，借用 Booch 的比喻，如同建造一座大厦和搭一个狗窝的差别。大型的、复杂的软件系统的开发是一项工程，必须按工程学的方法组织软件的生产与管理，必须经过计划、分析、设计、编程、测试、维护等一系列的软件生命周期阶段。这是人们从软件危机中获得的最重要的教益，这一认识促使了软件工程学的诞生。

软件工程学就是研究如何有效地组织和管理软件开发的工程学科。IEEE 在 1983 年将软件工程定义为：软件工程是开发、运行、维护和修复软件的系统方法。

著名的软件工程专家 Boehm 于 1983 年提出了软件工程的 7 条基本原理：

- 用分阶段的生命周期计划严格管理。
- 坚持进行阶段评审。
- 实行严格的产品控制。
- 采用现代程序设计技术。
- 结果应能清楚地审查。
- 开发小组的人员应该少而精。
- 承认不断改进软件工程实践的必要性。

软件工程方法学包含三个要素：方法、工具和过程。方法是指完成软件开发的各项任务的技术方法；工具是指为运用方法而提供的软件工程支撑环境；过程是指为获得高质量的软件所需要完成的一系列任务的框架。

近 30 年来，影响力最大、使用最广泛的软件工程方法学是结构化方法学和面向对象的方法学。

4.1.2 软件生命周期

软件生命周期（Software Life Cycle）是人们在研究软件开发过程时所发现的一种规律性的事实。如同人的一生要经历婴儿期、少年期、青年期、老年期直至死亡这样一个全过程一样，一个软件产品也要经历计划、分析、设计、编程、测试、维护直至被淘汰这样一个全过程。软件的这一全过程称为软件生命周期。

目前，软件生命周期各阶段的划分尚不统一，有的分得粗些，有的分得细些，所包含的实际内容也不完全相同。

- 1970年，Boehm提出了如表4-1所示的软件生命周期模型。

表 4-1 Boehm 定义的软件生命周期模型

计 划 时 期		开 发 时 期					运 行 时 期
问题定义	可行性研究	需求分析	总体设计	详细设计	编码	测试	维护

- 在1988年制定和公布的国家标准《GB 8566－1988计算机软件开发规范》中将软件生命周期划分为如表4-2所示的8个阶段。

表 4-2　GB8566 定义的软件生命周期模型

可行性研究与计划	需求分析	概要设计	详细设计	实现	组装测试	确认测试	使用和维护

- 在20世纪90年代初有了软件工程过程的概念之后，于1995年制定和公布的国家标准《GB/T 8566－1995信息技术－软件生存期过程》定义了软件生命周期的7个主要过程，如表4-3所示。

表 4-3　GB/T8566 定义的软件生命周期模型

<table>
<tr><td colspan="5">管　理　过　程</td></tr>
<tr><td>获取过程</td><td>供应过程</td><td>开发过程</td><td>运行过程</td><td>维护过程</td></tr>
<tr><td colspan="5">支持过程</td></tr>
</table>

其中，“获取过程”和“供应过程”分别描述了软件的“获取者”（用户）和“供应者”（开发者）在开发之前的主要活动和任务。而“管理过程”和“支持过程”则贯穿于整个软件生命周期。

- 1995年国际标准化组织对软件生命周期过程做了调整，公布了新的国际标准《ISO/IEC 12207信息技术－软件生存期过程》。该标准全面、系统地阐述了软件生命周期的3组共17个过程活动和任务，如表4-4所示。

表 4-4　ISO/IEC 定义的软件生命周期模型

主要过程	获取过程、供应过程、开发过程、运行过程、维护过程
支持过程	文档编制过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审核过程、问题解决过程
组织过程	管理过程、基础设施过程、改进过程、培训过程

- 1999年，Rational软件公司的3位软件工程大师Ivar Jacobson、Grady Booch和James Rumbaugh联合编写了一部划时代的著作《统一软件开发过程》（The Unified Software Development Process），将他们多年研究所得的软件开发方法学融合在了一起。该书清楚地说明了支持整个软件生命周期的统一软件开发过程是一个用例驱动的、以架构为中心的、迭代与增量的开发过程。统一软件开发过程是在重复一系列组成软件生命周期的循环，每次循环都包括如下的4个阶段和5种工作流，分别如表4-5和表4-6所示。

表 4-5　RUP 定义的软件生命周期模型的 4 个阶段

初始阶段（Inception Phase）	捕捉用例，思考系统架构……
细化阶段（Elaboration Phase）	细化用例，设计系统架构……
构造阶段（Construction Phase）	程序设计，实现，α 测试……
移交阶段（Transition Phase）	β 测试……

表 4-6　RUP 定义的软件生命周期模型的 5 种工作流

需求工作流（Requirements Workflow）	捕捉需求，使开发导向正确的系统
分析工作流（Analysis Workflow）	生成一个有助于架构设计的需求描述
设计工作流（Design Workflow）	建立系统设计模型
实现工作流（Implementation Workflow）	实现系统
测试工作流（Test Workflow）	验证实现的结果

尽管软件生命周期中各阶段的划分标准不统一，名称也不一致，但主要包括计划、分析、设计、编程、测试和维护等阶段。本书主要介绍分析、设计、测试和维护阶段的工作。

4.1.3　软件开发模型

为了指导软件的开发，可以用不同的方式将软件生命周期中的所有开发活动组织起来，从而形成了不同的软件开发模型。常见的开发模型有瀑布模型（Waterfall Model）、快速原型模型（Rapid Prototype Model）、演化模型（Evolutionary Model）、增量模型（Incremental Model）、螺旋模型（Spiral Model）和喷泉模型（Water Fountain model）等。

1．瀑布模型

瀑布模型严格遵循软件生命周期各阶段的固定顺序：计划、分析、设计、编程、测试和维护，上一阶段完成后才能进入到下一阶段，整个模型就像一个飞流直下的瀑布，如图 4-1 所示。

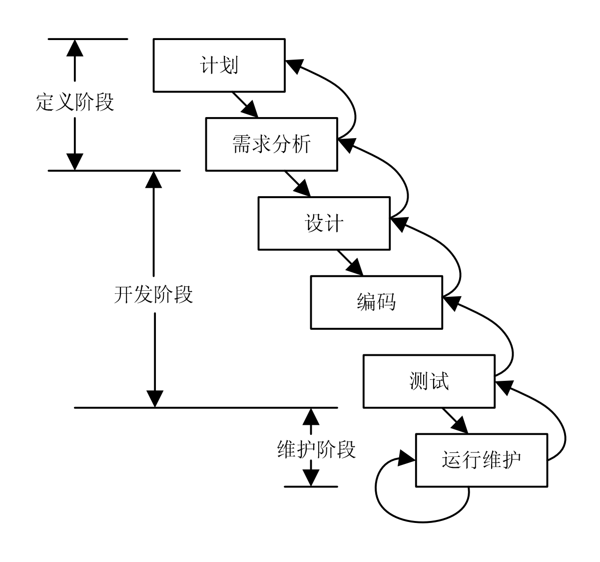


图 4-1　瀑布模型示意图

瀑布模型有许多优点：可强迫开发人员采用规范的方法；严格规定了各阶段必须提交的文档；要求每个阶段结束后，都要进行严格的评审。

但瀑布模型过于理想化，而且缺乏灵活性，无法在开发过程中逐渐明确用户难以确切表达或一时难以想到的需求，直到软件开发完成之后才发现与用户需求有很大距离，此时必须付出高额的代价才能纠正这一偏差。

2．快速原型模型

快速原型是指快速建立起来的可以在计算机上运行的程序，它所完成的功能往往是最终软件产品功能的一个子集。快速原型模型的第一步是快速建立一个能反映用户主要需求的软件原型，让用户在计算机上使用它，通过实际操作了解目标系统的概貌。开发人员按照用户提出的意见快速地修改原型系统，然后再次请用户试用……一旦用户认为这个原型

系统确实能够满足他们的需求，开发人员便可据此书写软件需求说明，并根据这份文档开发出可以满足用户真实需求的软件产品。

原型化方法基于这样一种客观事实：并非所有的需求在系统开发之前都能准确地说明和定义。因此，它不追求也不可能要求对需求的严格定义，而是采用了动态定义需求的方法。

具有广泛技能高水平的原型化人员是原型实施的重要保证。原型化人员应该是具有经验与才干、训练有素的专业人员。衡量原型化人员能力的重要标准是他是否能够从用户的模糊描述中快速获取实际的需求。

3．演化模型

演化模型也是一种原型化开发方法，但与快速原型模型略有不同。在快速原型模型中，原型的用途是获知用户的真正需求，一旦需求确定了，原型即被抛弃。而演化模型的开发过程，则是从初始模型逐步演化为最终软件产品的渐进过程。也就是说，快速原型模型是一种“抛弃式”的原型化方法，而演化模型则是一种“渐进式”的原型化方法。

4．增量模型

增量模型是第三种原型化开发方法，但它既非“抛弃式”的，也非“渐进式”的，而是“递增式”的。增量模型把软件产品划分为一系列的增量构件，分别进行设计、编程、集成和测试。每个构件由多个相互作用的模块构成，并且能够完成特定的功能。如何将一个完整软件产品分解成增量构件，因软件产品特点和开发人员的习惯而异，但使用增量模型的软件体系结构必须是开放的，加入新构件的过程必须简单方便，新的增量构件不得破坏已经开发出来的产品。其示意图如图 4-2 所示。

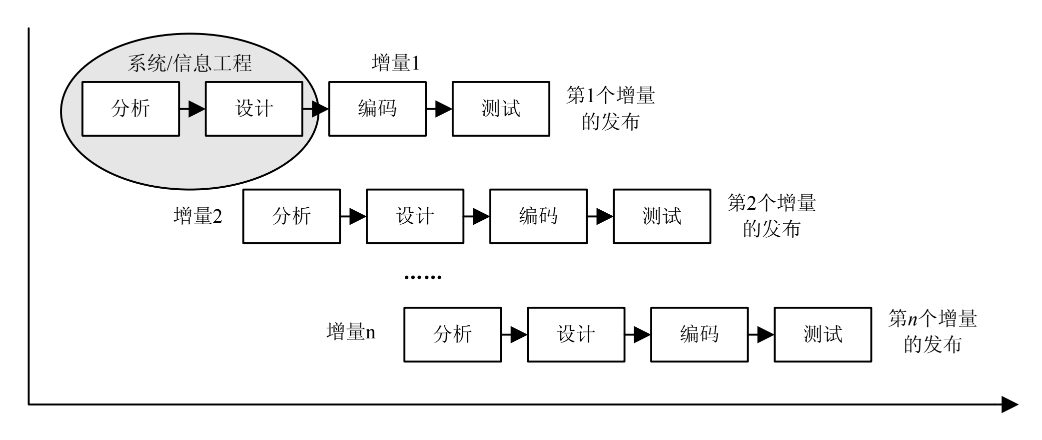


图 4-2　增量模型示意图

5．螺旋模型

螺旋模型综合了瀑布模型和演化模型的优点，还增加了风险分析。螺旋模型包含 4 个方面的活动：制订计划、风险分析、实施工程、客户评估。这 4 项活动恰好可以放在一个直角坐标系的 4 个象限，而开发过程恰好像一条螺旋线。采用螺旋模型时，软件开发沿着螺旋线自内向外旋转，每转一圈都要对风险进行识别和分析，并采取相应的对策。螺旋线第一圈的开始点可能是一个概念项目。从第二圈开始，一个新产品开发项目开始了，新产品的演化沿着螺旋线进行若干次迭代，一直运转到软件生命期结束。其示意图如图 4-3 所示。

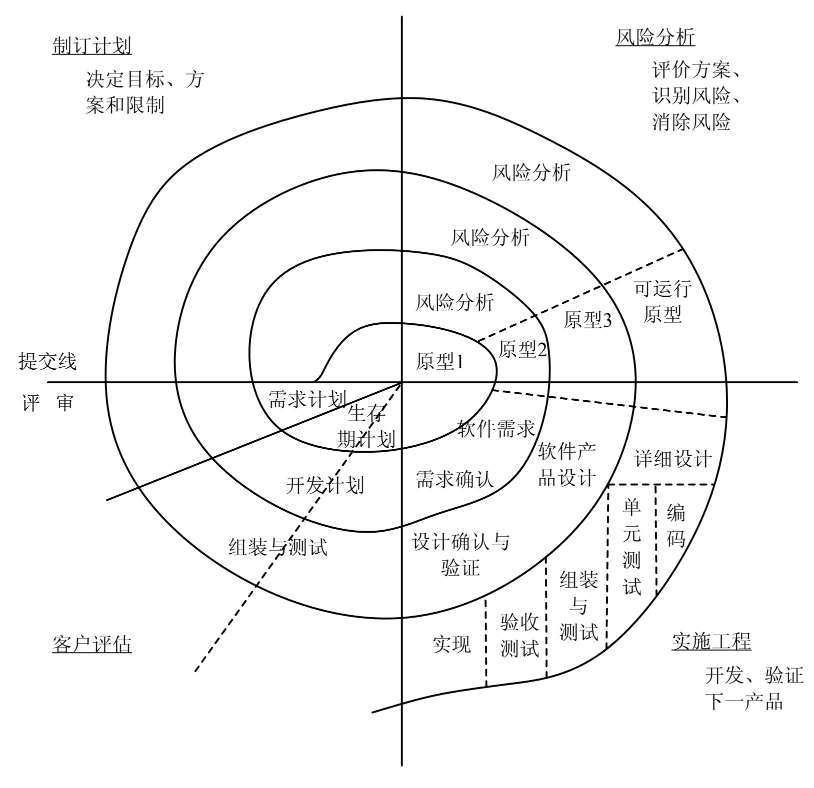


图 4-3 螺旋模型示意图

6．喷泉模型

喷泉模型主要用于描述面向对象的开发过程。喷泉一词体现了面向对象开发过程的迭代和无间隙特征。迭代意味着模型中的开发活动常常需要多次重复，每次重复都会增加或明确一些目标系统的性质，但却不是对先前工作结果的本质性改动。无间隙是指在开发活动（如分析、设计、编程）之间不存在明显的边界，而是允许各开发活动交叉、迭代地进行。

7．基于构件的模型

构件（Component，也称为组件）是一个具有可重用价值的、功能相对独立的软件单元。基于构件的软件开发（Component Based Software Development，CBSD）模型是利用模块化方法，将整个系统模块化，并在一定构件模型的支持下，复用构件库中的一个或多个软件构件，通过组合手段高效率、高质量地构造应用软件系统的过程。基于构件的开发模型融合了螺旋模型的许多特征，本质上是演化型的，开发过程是迭代的。基于构件的开发模型由软件的需求分析和定义、体系结构设计、构件库建立、应用软件构建以及测试和发布 5 个阶段组成。采用基于构件的开发模型的软件过程如图 4-4 所示。

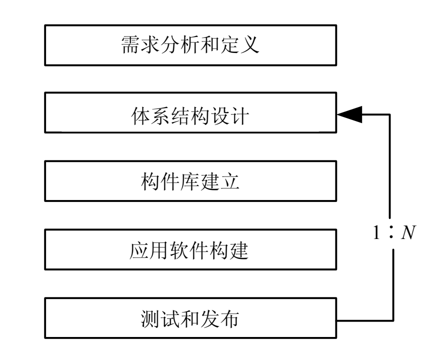


图 4-4 采用基于构件的开发模型的软件过程

构件作为重要的软件技术和工具得到了极大的发展，这些新技术和工具有 Microsoft 的 DCOM、Sun 的 EJB、OMG 的 CORBA 等。基于构件的开发活动从标识候选构件开始，通过搜索已有构件库，确认所需要的构件是否已经存在，如果已经存在，就从构件库中提取出来复用；如果不存在，就采用面向对象方法开发它。在提取出来的构件通过语法和语义检查后，将这些构件通过胶合代码组装到一起实现系统，这个过程是迭代的。

基于构件的开发方法使得软件开发不再一切从头开始，开发的过程就是构件组装的过程，维护的过程就是构件升级、替换和扩充的过程，其优点是构件组装模型导致了软件的复用，提高了软件开发的效率；构件可由一方定义其规格说明，被另一方实现，然后供给第三方使用；构件组装模型允许多个项目同时开发，降低了费用，提高了可维护性，可实现分步提交软件产品。

其缺点是由于采用自定义的组装结构标准，缺乏通用的组装结构标准，引入具有较大的风险；可重用性和软件高效性不易协调，需要精干的、有经验的分析人员和开发人员，一般的开发人员插不上手，客户的满意度低；过分依赖于构件，构件库的质量影响着产品质量。

8．快速应用开发模型（RAD）

快速应用开发（Rapid Application Development，RAD）模型是一个增量型的软件开发过程模型，强调极短的开发周期。RAD 模型是瀑布模型的一个“高速”变种，通过大量使用可复用构件，采用基于构件的建造方法赢得快速开发。如果需求理解得好且约束了项目的范围，利用这种模型可以很快地创建出功能完善的“信息系统”。其流程从业务建模开始，随后是数据建模、过程建模、应用生成、测试及反复。采用 RAD 模型的开发过程如图 4-5 所示。

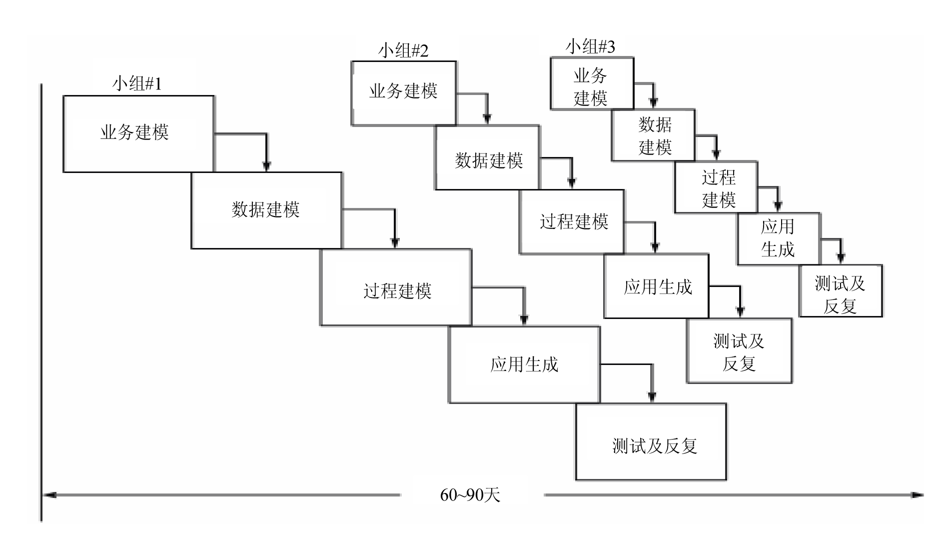


图 4-5　采用 RAD 模型的开发过程

RAD 模型各个活动期所要完成的任务如下。

- 业务建模：以什么信息驱动业务过程运作？要生成什么信息？谁生成它？信息流的去向是哪里？由谁处理？可以辅之以数据流图。
- 数据建模：为支持业务过程的数据流，找数据对象集合，定义数据对象属性，与其他数据对象的关系构成数据模型，可辅之以E-R图。

- 过程建模：使数据对象在信息流中完成各业务功能。创建过程以描述数据对象的增加、修改、删除、查找，即细化数据流图中的处理框。
- 应用程序生成：利用第四代语言（4GL）写出处理程序，重用已有构件或创建新的可重用构件，利用环境提供的工具自动生成并构造出整个应用系统。
- 测试与交付，由于大量重用，一般只做总体测试，但新创建的构件还是要测试的。

与瀑布模型相比，RAD 模型不采用传统的第三代程序设计语言来创建软件，而是采用基于构件的开发方法，复用已有的程序结构（如果可能）或使用可复用构件，或创建可复用的构件（如果需要）。在所有情况下，均使用自动化工具辅助软件创造。很显然，加在一个 RAD 模型项目上的时间约束需要“一个可伸缩的范围”。如果一个业务能够被模块化使得其中每一个主要功能均可以在不到三个月的时间内完成，那么它就是 RAD 的一个候选者。每一个主要功能可由一个单独的 RAD 组来实现，最后再集成起来形成一个整体。

RAD 模型通过大量使用可复用构件加快了开发速度，对信息系统的开发特别有效。但是像所有其他软件过程模型一样，RAD 方法也有其缺陷：

- 并非所有应用都适合RAD。RAD模型对模块化要求比较高，如果有哪一项功能不能被模块化，那么建造RAD所需要的构件就会有问题；如果高性能是一个指标，且该指标必须通过调整接口使其适应系统构件才能赢得，RAD方法也有可能不能奏效。
- 开发者和客户必须在很短的时间内完成一系列的需求分析，任何一方配合不当都会导致RAD项目失败。
- RAD只能用于信息系统开发，不适合技术风险很高的情况。当一个新应用要采用很多新技术或当新软件要求与已有的计算机程序有较高的互操作性时，这种情况就会发生。

9．RUP 方法

RUP（Rational Unified Process）是一个统一的软件开发过程，是一个通用过程框架，可以应付种类广泛的软件系统、不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。RUP 是基于构件的，这意味着利用它开发的软件系统是由构件构成的，构件之间通过定义良好的接口相互联系。在准备软件系统所有蓝图的时候，RUP 使用的是统一建模语言 UML。

与其他软件过程相比，RUP 具有 3 个显著的特点：用例驱动，以基本架构为中心，以及迭代和增量。

RUP 中的软件过程在时间上被分解为 4 个顺序的阶段，分别是初始阶段、细化阶段、构建阶段和交付阶段。每个阶段结束时都要安排一次技术评审，以确定这个阶段的目标是否已经满足。如果评审结果令人满意，就可以允许项目进入下一个阶段。基于 RUP 的软件过程模型如图 4-6 所示。

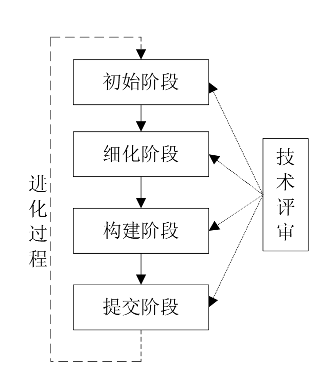


图 4-6　基于 RUP 的软件过程

从图 4-6 中可以看出：基于 RUP 的软件过程是一个迭代过程。完成初始、细化、构建和提交 4 个阶段就是一个开发周期，每次经过这 4 个阶段就会

产生一代软件。除非产品退役，否则通过重复同样的 4 个阶段，产品将演化为下一代产品，但每一次的侧重点都将放在不同的阶段上。这些随后的过程称为演化过程。

在进度和工作量方面，所有阶段都各不相同。尽管不同的项目有很大的不同，但一个中等规模项目的典型初始开发周期应该预先考虑到工作量和进度间的分配，如表 4-7 所示。

表 4-7 RUP 各阶段的工作量和进度分配

	初始阶段	细化阶段	构建阶段	提交阶段
工作量	5%	20%	65%	10%
进度	10%	30%	50%	10%

对于演进周期，初始和细化阶段就小得多了。能够自动完成某些构建工作的工具将会缓解此现象，并使得构建阶段比初始阶段和细化阶段的总和还要小很多。

RUP 的工作流程分为两部分：核心工作流程与核心支持工作流程。核心工作流程（在项目中的流程）包括业务需求建模、分析设计、实施、测试、部署；核心支持工作流程（在组织中的流程）包括环境、项目管理、配置与变更管理。

统一过程 4 个阶段的核心任务，以及需要提交的文档和模型分别如下。

1）*初始阶段*

①核心任务

- 明确地说明项目规模。这涉及了解环境及最重要的需求和约束，以便于可以得出最终产品的验收标准。
- 计划和准备商业理由。评估风险管理、人员配备、项目计划和成本/进度/收益率折中的备选方案。
- 综合考虑备选构架，评估设计和自制/外购/复用方面的折中，从而估算出成本、进度和资源。此处的目标在于通过对一些概念的证实来证明可行性。该证明可采用可模拟需求的模型形式或用于探索被认为高风险区域的初始原型。先启阶段的原型设计工作应该限制在确信解决方案可行就可以。该解决方案在精化和构建阶段实现。
- 准备项目的环境，评估项目和组织，选择工具，决定流程中要改进的部分。

②需要提交的文档和模型（见表 4-8）

表 4-8 初始阶段需要提交的文档和模型

核心文档及模型	里程碑状态
前景	已经对核心项目的需求、关键功能和主要约束进行了记录
商业理由	已经确定并得到了批准
风险列表	已经确定了最初的项目风险
软件开发计划	已经确定了最初阶段及其持续时间和目标。软件开发计划中的资源估算（特别是时间、人员和开发环境成本）必须与商业理由一致。资源估算可以涵盖整个项目直到交付所需的资源，也可以只包括进行精化阶段所需的资源。此时，整个项目所需的资源估算应该看作大致的“粗略估计”。该估算在每个阶段和每次迭代中都会更新，并且随着每次迭代变得更加准确。根据项目的需要，可能在某种条件下完成了一个或多个附带的“计划”工件。此外，附带的“指南”工件通常也至少完成了“草稿”
迭代计划	第一个精化迭代的迭代计划已经完成并经过了复审

续表

核心文档及模型	里程碑状态
软件验收计划	完成复审并确定了基线；随着其他需求的发现，将对其在随后的迭代中进行改进
项目专用模板	已使用文档模板制作了文档工件
用例建模指南	确定了基线
工具	选择了支持项目的所有工具。安装了对先启阶段的工作必要的工具
词汇表	已经定义了重要的术语；完成了词汇表的复审
用例模型	已经确定了重要的主角和用例，只为最关键的用例简要说明了事件流
领域模型	已经对系统中使用的核心概念进行了记录和复审。在核心概念之间存在特定关系的情况下，已用作对词汇表的补充
原型	概念原型的一个或多个证据，以支持前景和商业理由，解决非常具体的风险

2）细化阶段

①核心任务

- 快速确定构架，确认构架并为构架建立基线。
- 根据此阶段获得的新信息改进前景，对推动构架和计划决策的最关键用例建立可靠的了解。
- 为构建阶段创建详细的迭代计划并为其建立基线。
- 改进开发案例，定位开发环境，包括流程和支持构建团队所需的工具和自动化支持。
- 改进构架并选择构件。评估潜在构件，充分了解自制/外购/复用决策，以便有把握地确定构建阶段的成本和进度。集成了所选构架构件，并按主要场景进行了评估。通过这些活动得到的经验有可能导致重新设计构架、考虑替代设计或重新考虑需求。

②需要提交的文档和模型（见表 4-9）

表 4-9　细化阶段需要提交的文档和模型

核心文档及模型	里程碑状态
原型	已经创建了一个或多个可执行构架原型，以探索关键功能和构架上的重要场景
风险列表	已经进行了更新和复审。新的风险可能是构架方面的，主要与处理非功能性需求有关
项目专用模板	已使用文档模板制作了文档工件
工具	已经安装了用于支持精化阶段工作的工具
软件构架文档	编写完成并确定了基线，如果系统是分布式的或必须处理并行问题，则包括构架上重要用例的详细说明（用例视图）、关键机制和设计元素的标识（逻辑视图），以及（部署模型的）进程视图和部署视图的定义
设计模型（和所有组成部分）	制作完成并确定了基线。已经定义了构架方面重要场景的用例实现，并将所需行为分配给了适当的设计元素。已经确定了构件并充分理解了自制/外购/复用决策，以便有把握地确定构建阶段的成本和进度。集成了所选构架构件，并按主要场景进行了评估。通过这些活动得到的经验有可能导致重新设计构架、考虑替代设计或重新考虑需求
数据模型	制作完成并确定了基线。已经确定并复审了主要的数据模型元素（如重要实体、关系和表）
实施模型（以及所有组成工件，包括构件）	已经创建了最初结构，确定了主要构件并设计了原型
前景	已经根据此阶段获得的新信息进行了改进，对推动构架和计划决策的最关键用例建立了可靠的了解

续表

核心文档及模型	里程碑状态
软件开发计划	已经进行了更新和扩展，以便涵盖构建阶段和产品化阶段
指南，如设计指南和编程指南	使用指南对工作进行了支持
迭代计划	已经完成并复审了构建阶段的迭代计划
用例模型	用例模型（大约完成 80%）——已经在用例模型调查中确定了所有用例、确定了所有主角并编写了大部分用例说明（需求分析）
补充规约	已经对包括非功能性需求在内的补充需求进行了记录和复审
可选	里程碑状态
商业理由	如果构架调查不涵盖变更基本项目假设的问题，则已经对商业理由进行了更新
分析模型	可能作为正式工件进行了开发；进行了经常但不正式的维护，正演进为设计模型的早期版本
培训材料	用户手册与其他培训材料，根据用例进行了初步起草。如果系统具有复杂的用户界面，可能需要培训材料

3）构建阶段

①核心任务

- 资源管理、控制和流程优化。
- 完成构件开发并根据已定义的评估标准进行测试。
- 根据前景的验收标准对产品发布版进行评估。

②需要提交的文档和模型（见表 4-10）

表 4-10　构建阶段需要提交的文档和模型

核心文档及模型	里程碑状态
“系统”	可执行系统本身随时可以进行“Beta”测试
部署计划	已开发最初版本，进行了复审并建立了基线
实施模型	对在精化阶段创建的模型进行了扩展；构建阶段末期完成所有构件的创建
测试模型	为验证构建阶段所创建的可执行发布版而设计并开发的测试
培训材料	用户手册与其他培训材料，根据用例进行了初步起草。 如果系统具有复杂的用户界面，可能需要培训材料
迭代计划	已经完成并复审了产品化阶段的迭代计划
设计模型	已经用新设计元素进行了更新，这些设计元素是在完成所有需求期间确定的
项目专用模板	已使用文档模板制作了文档工件
工具	已经安装了用于支持构建阶段工作的工具
数据模型	已经用支持持续实施所需的所有元素（如表、索引、对象关系型映射等）进行了更新
可选	里程碑状态
补充规约	已经用构建阶段发现的新需求（如果有）进行了更新
用例模型	已经用构建阶段发现的新用例（如果有）进行了更新

4）产品化阶段（提交阶段）

①核心任务

- 执行部署计划。

- 对最终用户支持材料定稿。
- 在开发现场测试可交付产品。
- 制作产品发布版。
- 获得用户反馈。
- 基于反馈调整产品。
- 使最终用户可以使用产品。

②需要提交的文档和模型（见表 4-11）

表 4-11　提交阶段需要提交的文档和模型

核心文档及模型	里程碑状态
产品工作版本	已按照产品需求完成，客户应该可以使用最终产品
发布说明	完成
安装产品与模型	完成
培训材料	完成，以确保客户自己可以使用和维护产品
最终用户支持材料	完成，以确保客户自己可以使用和维护产品
可选	里程碑状态
测试模型	在客户想要进行现场测试的情况下，可以提供测试模型

10．XP 方法

敏捷方法中最著名的就是 XP、XP 是一种轻量、高效、低风险、柔性、可预测、科学且充满乐趣的软件开发方式，适用于小型或中型软件开发团队，并且客户的需求模糊或需求多变。与其他方法相比，其最大的不同如下：

（1）在更短的周期内，更早地提供具体、持续的反馈信息。

（2）迭代地进行计划编制，首先在最开始迅速生成一个总体计划，然后在整个项目开发过程中不断地发展它。

（3）依赖于自动测试程序来监控开发进度，并及早地捕获缺陷。

（4）依赖于口头交流、测试和源程序进行沟通。

（5）倡导持续的演化式的设计。

（6）依赖于开发团队内部的紧密协作。

（7）尽可能达到程序员短期利益和项目长期利益的平衡。

XP 由价值观、原则、实践和行为 4 个部分组成，它们彼此相互依赖、关联，并通过行为贯穿于整个生命周期。XP 的核心是其总结的四大价值观，即沟通、简单、反馈和勇气。它们是 XP 的基础，也是 XP 的灵魂。XP 的 5 个原则是快速反馈、简单性假设、逐步修改、提倡更改和优质工作。而在 XP 方法中，贯彻的是“小步快走”的开发原则，因此工作质量决不可打折扣，通常采用测试先行的编码方式来提供支持。

在 XP 中，集成了 12 个最佳实践，分别是计划游戏、小型发布、隐喻、简单设计、测试先行、重构、结对编程、集体代码所有制、持续集成、每周工作 40 小时、现场客户和编码标准。当然，这些所谓的“最佳实践”并非对每个项目都是最佳的，需要项目团队根据实际情况决定。而且，XP 方法的有些原则在应用中不一定能得到贯彻和执行。因此，在实际工作中，应该“取其精华，去其糟粕”，把 XP 方法和其他方法结合起来。

4.2 主要软件开发方法

本节将介绍主要软件开发方法。

4.2.1 结构化分析和设计

1977年出现的结构化方法学也称为生命周期方法学，它采用结构化技术（结构化分析、结构化设计和结构化实现）来完成软件开发的各项任务。这种方法学把软件生命周期的全过程依次划分为若干个阶段，然后顺序地完成每个阶段的任务。

结构化方法学具有如下特点。

- 阶段性。前一阶段工作完成以后，后一阶段工作才能开始，前一阶段的输出文档是后一阶段的输入文档。
- 推迟实施。将分析和设计阶段与实施分开，适当地推迟系统的具体程序实现。
- 文档管理。在每一阶段都规定了要完成的文档资料，没有完成文档，就认为没有完成该阶段的任务。在每一阶段都要对已完成的文档进行复审，以便尽早发现问题，避免后期的返工。

把软件生命周期划分成若干个阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发过程的困难程度；在每个阶段结束之前都要从技术和管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发的全过程以一种有条不紊的方式进行。

结构化方法学曾经给软件产业带来巨大进步，在一定程度上缓解了软件危机。但结构化方法基于功能分析与功能分解，软件结构紧密依赖于系统功能，而在实际开发工作中，系统功能往往又是模糊易变的，功能的变化经常会引起软件结构的整体修改。

目前，面向对象方法已取代结构化方法成为软件方法学的主流。但对于一些功能需求非常明确而且不会轻易改变的软件系统，结构化方法仍然比较有效。

1．结构化分析

结构化分析（Structured Analysis，SA）方法是一种面向数据流的需求分析方法。它的基本思想是自顶向下逐层分解，把一个大问题分解成若干个小问题，每个小问题再分解成若干个更小的问题。经过逐层分解，每个最低层的问题都是足够简单、容易解决的，于是复杂的问题也将迎刃而解。

数据流图和数据字典是结构化分析的常见工具，软件需求说明书是需求分析阶段的最后成果。

1）数据流图

数据流图（Data Flow Diagram，DFD）用来描述数据流从输入到输出的变换流程。关于更多详细的内容思考可参见“数据流图设计”的内容。

2）数据字典

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合。

数据流图和数据字典共同构成系统的逻辑模型。没有数据流图，数据字典难以发挥作用；没有数据字典，数据流图就不严格。只有把数据流图和对数据流图中每个元素的精确

定义放在一起，才能共同构成系统的规格说明。关于更多详细的内容可参见“数据设计”的内容。

2．结构化设计

系统设计是软件生命周期的重要组成部分，主要包括体系结构设计、接口设计、数据设计和过程设计。

结构化设计（Structured Design，SD）方法是一种面向数据流的设计方法，它是以结构化分析阶段所产生的文档（包括数据流图、数据字典和软件需求说明书）为基础，自顶向下，逐步求精和模块化的过程。结构化设计通常可分为概要设计和详细设计。概要设计的任务是确定软件系统的结构，进行模块划分，确定每个模块的功能、接口及模块间的调用关系。详细设计的任务是为每个模块设计实现的细节。更多详细的内容可参见“软件设计概述”的内容。

1）概要设计

经过需求分析阶段的工作，系统必须已经清楚了“做什么”，概要设计的基本目的就是回答“概括地说，系统应该如何实现？”这个问题。概要设计的重要任务就是设计软件的结构，也就是要确定系统是由哪些模块组成的，以及这些模块相互间的关系。

SD 方法采用结构图（Structure Chart）来描述程序的结构。构成程序结构图的主要成分有模块、调用和数据，结构图中的模块用矩形表示，在矩形框内可标上模块的名字。模块间如有箭头或直线相连，表明它们之间有调用关系。SD 方法有时也使用层次图和 HIPO 图（层次图加输入/处理/输出图）。

整个概要设计过程主要包括如下内容。

第 1 步：复查基本系统模型。复查的目的是确保系统的输入数据和输出数据符合实际。

第 2 步：复查并精化数据流图。应该对需求分析阶段得到的数据流图认真复查，并且在必要时进行精化。不仅要确保数据流图给出了目标系统的正确的逻辑模型，而且应该使数据流图中每个处理都代表一个规模适中、相对独立的子功能。

第 3 步：确定数据流图的信息流类型。数据流图中从系统的输入数据流到系统的输出数据流的一连串连续变换形成了一条信息流。信息流大体可分为两种类型。

- 变换流：信息沿着输入通道进入系统，然后通过变换中心（也称为主加工）处理，再沿着输出通道离开系统。具有这一特性的信息流称为变换流。具有变换流型的数据流图可明显地分成输入、变换（主加工）、输出三大部分。
- 事务流：信息沿着输入通道到达一个事务中心，事务中心根据输入信息（即事务）的类型在若干个动作序列（称为活动流）中选择一个来执行，这种信息流称为事务流。事务流有明显的事务中心，各活动以事务中心为起点呈辐射状流出。

第 4 步：根据流类型分别实施变换分析或事务分析。变换分析是从变换流型的数据流图导出程序结构图。具体过程如下。

- 确定输入流和输出流的边界，从而孤立出变换中心。
- 完成第一级分解，设计模块结构的顶层和第一层。
- 完成第二级分解，也就是输入控制模块、变换控制模块和输出控制模块的分解，设计中、下层模块。

事务分析是从事务流型的数据流图导出程序结构图，具体过程如下：

- 确定事务中心和每条活动流的流特性。
- 将事务流型数据流图映射成高层的程序结构，分解出接收模块、发送模块（调度模块），以及发送模块所控制的下层所有的活动流模块。
- 进一步完成接收模块和每一个活动流模块的分解。

第 5 步：根据软件设计原则对得到的软件结构图进一步优化。

2）详细设计

概要设计已经确定了每个模块的功能和接口，详细设计的任务就是为每个模块设计其实现的细节。详细设计阶段的根本目标是确定应该怎样具体地实现所要求的系统，得出对目标系统的精确描述。

结构化程序设计（Structured Programming，SP）采用自顶向下逐步求精的设计方法和单入口、单出口的控制结构。在设计一个模块的实现算法时先考虑整体后考虑局部，先抽象后具体，通过逐步细化，最后得到详细的实现算法。单入口、单出口的控制结构使程序的静态结构和动态执行过程一致，具有良好的结构，增强了程序的可读性。

针对在程序中大量无节制地使用 GOTO 语句而导致程序结构混乱的现象，Dijkstra 于 1965 年提出在程序语言中取消 GOTO 语句。1966 年，Bohm 和 Jacopini 证明了任何单入口、单出口、没有死循环的程序都能用 3 种基本的控制结构来构造，这 3 种基本的控制结构是：顺序结构、IF_THEN_ELSE 型分支结构（选择结构）和 DO_WHILE 型循环结构。如果程序设计中只允许使用这三种基本的控制结构，则称为经典的结构化程序设计；如果还允许使用 DO_CASE 型多分支结构和 DO_UNTIL 型循环结构，则称为扩展的结构化程序设计；如果再加上允许使用 LEAVE（或 BREAK）结构，则称为修正的结构化程序设计。

应用于详细设计的工具主要包括如下几种。

- 程序流程图：又称为程序框图，它是历史最悠久的描述过程设计的方法，然而它也是用得最混乱的一种方法。程序流程图的主要优点是对控制流程的描绘很直观，便于初学者掌握。但由于程序流程图中用箭头代表控制流，经常诱使程序员不顾结构化程序设计的精神而随意转移控制，且不支持逐步求精方法，不易表现数据结构。程序流程图尽管有种种缺点，许多人建议停止使用它，但至今仍在广泛使用着。不过总的趋势是越来越多的人不再使用程序流程图。
- 盒图（N－S图）：盒图是由Nassi和Shneiderman提出的一种符合结构化设计原则的图形描述工具，它仅含5种基本的控制结构，顺序结构、IF-THEN-ELSE型分支结构、CASE型多分支结构、DO-WHILE和DO-UNTIL型循环结构、子程序结构。盒图具有如下特点。
  - 功能域（即一个特定控制结构的作用域）明确，可以从盒图上一眼就看出来。
  - 由于没有箭头，不可能任意转移控制。
  - 容易确定局部和全程数据的作用域。
  - 容易表示嵌套关系，也可以表示模块的层次结构。

坚持使用盒图作为详细设计的工具，可以使程序员逐步养成用结构化的方式思考问题和解决问题的习惯。

- PAD图：问题分析图（Problem Analysis Diagram）的英文缩写，它用二维树型结构的图表示程序的控制流，比较容易翻译成机器代码。PAD图具有如下特点。
  - 使用表示结构化控制结构的PAD符号所设计出来的程序必然是结构化程序。
  - PAD图所描绘的程序结构十分清晰。
  - 用PAD图表现程序逻辑，易读、易懂、易记。
  - 容易将PAD图转换成高级语言源程序，这种转换可用软件工具自动完成。
  - PAD图既可表示程序逻辑，也可用于描绘数据结构。
  - PAD图的符号支持自顶向下、逐步求精方法的使用。
- PDL：程序设计语言（Program Design Language）的英文缩写，也称为伪码，是一种以文本方式表示数据和处理过程的设计工具。PDL是一种非形式化语言，它对控制结构的描述是确定的，但控制结构内部的描述语法是不确定的，它可根据不同的应用领域和不同的设计层次灵活选用其描述方式，甚至可用自然语言描述。与程序语言（Programming Language）不同，PDL程序是不可执行的，但它可以通过转换程序自动转换成某种高级程序语言的源程序。

常见的详细设计工具还包括判定树、判定表等。

4.2.2 面向数据结构的设计

前面讲的结构化设计方法是面向数据流的，另外还有一种面向数据结构的设计方法。它根据输入/输出数据结构导出程序结构。

在许多应用领域中信息都有清楚的层次结构，输入数据、内部存储的信息（数据库或文件）及输出数据都可能有独特的结构。数据结构既影响程序的结构，又影响程序的处理，重复出现的数据通常由具有循环控制结构的程序来处理，选择数据（即可能出现也可能不出现的信息）要用带有分支控制的程序来处理。层次的数据组织通常和使用这些数据的程序的层次结构十分相似。面向数据结构设计方法的基本思想就是根据数据结构导出程序结构。

Jackson 方法和 Warnier 方法是最著名的两种面向数据结构的设计方法。

Jackson 方法的基本步骤是：建立系统的数据结构；以数据结构为基础，对应地建立程序结构；列出程序中要用到的各种基本操作，再将这些操作分配到程序结构适当的模块中。

对于 Warnier 方法，这里不再详细介绍。

由于面向数据结构的设计方法并不明显地使用软件结构的概念，对于模块独立原则也没有给予应有的重视，因此并不适合于复杂的软件系统。

4.2.3 面向对象的分析与设计

结构化分析和设计方法在一定程度上缓解了“软件危机”。但随着人们对软件提出的要求越来越高，结构化方法已经无法承担快速、高效地开发复杂软件系统的重任。20 世纪 80 年代逐渐成熟的面向对象方法学，使软件开发者对软件的分析、设计和编程等方面都有了全新的认识。由于“对象”概念的引入，将数据和方法封装在一起，提高了模块的聚合度，降低了模块的耦合度，更大程度上支持了软件重用，从而十分有效地降低了软件的复杂度，提高了软件开发的生产率。目前，面向对象方法学已成为软件开发者的第一选择。

1．面向对象方法学概述

究竟怎样才算真正的“面向对象”（Object-Oriented，OO）？Peter Coad 和 Edward Yourdon 提出了下列等式：

面向对象 ＝ 对象（Objects）

＋ 类（Classes）

＋ 继承（Inheritance）

＋ 消息通信（Communication With Messages）

1）对象与封装

对象（Object）是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。面向对象的软件系统是由对象组成的，复杂的对象由比较简单的对象组合而成。也就是说，面向对象方法学使用对象分解取代了传统方法的功能分解。

对象三要素包括对象标识、属性和服务。

对象标识（Object Identifier），也就是对象的名字，供系统内部唯一地识别对象。定义或使用对象时，均应指定对象标识。

属性（Attribute），也称为状态（State）或数据（Data），用来描述对象的静态特征。在某些面向对象的程序设计语言中，属性通常被称为成员变量（Member Variable）或简称变量（Variable）。

服务（Service），也称为操作（Operation）、行为（Behavior）或方法（Method）等，用来描述对象的动态特征。在某些面向对象的程序设计语言中，服务通常被称为成员函数（Member Function）或简称函数（Function）。

封装（Encapsulation）是对象的一个重要原则。它有两层含义：第一，对象是其全部属性和全部服务紧密结合而形成的一个不可分割的整体；第二，对象是一个不透明的黑盒子，表示对象状态的数据和实现操作的代码都被封装在黑盒子里面。使用一个对象的时候，只需知道它向外界提供的接口形式，无须知道它的数据结构细节和实现操作的算法。从外面看不见，也就更不可能从外面直接修改对象的私有属性。

2）类

类（Class）是对具有相同属性和服务的一个或一组对象的抽象定义。

类与对象是抽象描述与具体实例的关系，一个具体的对象被称作类的一个实例（Instance）。

3）继承与多态性

继承（Inheritance）是面向对象方法学中的一个十分重要的概念，其定义是：特殊类（或称子类、派生类）的对象拥有其一般类（或称父类、基类）的全部属性与服务，称作特殊类对一般类的继承。在面向对象的方法学中，继承是提高软件开发效率的重要原因之一。

多态性（Polymorphism）是指一般类中定义的属性或服务被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为。使用多态技术时，用户可以发送一个通用的消息，而实现的细节则由接受对象自行决定，这样同一消息就可以调用不同的方法。多态性不仅增加了面向对象软件系统的灵活性，进一步减少了信息冗余，而且显著提高了软件的可重

用性和可扩充性。多态有多种不同的形式，其中参数多态和包含多态称为通用多态，过载多态和强制多态称为特定多态。

4）消息通信

消息（Message）就是向对象发出的服务请求，它应该含有下述信息：提供服务的对象标识、消息名、输入信息和回答信息。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，相反，它是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。

消息通信（Communication With Messages）也是面向对象方法学中的一条重要原则，它与对象的封装原则密不可分。封装使对象成为一些各司其职、互不干扰的独立单位；消息通信则为它们提供了唯一合法的动态联系途径，使它们的行为能够互相配合，构成一个有机的系统。

只有同时使用对象、类、继承与消息通信，才是真正面向对象的方法。

5）面向对象方法学的优点

- 与人类习惯的思维方法一致：面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使软件开发的方法与过程尽可能接近人类认识世界解决问题的方法与过程，也就是使描述问题的“问题域”与解决问题的“解域”在结构上尽可能一致。
- 稳定性好：传统的软件开发方法基于功能分析与功能分解，软件结构紧密依赖于系统所要完成的功能，当功能需求发生变化时将引起软件结构的整体修改。而用户需求变化大部分是针对功能的，因此这样的系统是不稳定的。

面向对象的方法用对象模拟问题域中的实体，以对象为中心构造软件系统，系统的功能需求变化时并不会引起软件结构的整体变化。由于现实世界中的实体是相对稳定的，因此以对象为中心构造的软件系统也是比较稳定的。

- 可重用性好：面向对象方法学在利用可重用的软件成分构造新的软件系统时有很大的灵活性。继承机制与多态性使得子类不仅可以重用其父类的数据结构与程序代码，并且可以方便地修改和扩充，而这种修改并不影响对原有类的使用。
- 较易开发大型软件产品：用面向对象方法学开发软件时，构成软件系统的每个对象相对独立。因此，可以把一个大型软件产品分解成一系列相互独立的小产品来处理。这不仅降低了开发的技术难度，而且也使得对开发工作的管理变得容易多了。
- 可维护性好：面向对象的软件比较容易理解、容易修改、容易测试。

2．面向对象的分析

综观计算机软件发展史，许多新方法和新技术都是在编程领域首先兴起，进而发展到软件生命周期的前期阶段——分析与设计阶段。结构化方法经历了从“结构化编程”、“结构化设计”到“结构化分析”的发展历程，面向对象的方法也经历了从“面向对象的编程”（Object-Oriented Programming，OOP）、“面向对象的设计”（Object-Oriented Design，OOD）到“面向对象的分析”（Object-Oriented Analysis，OOA）的发展历程。1989 年之后，面向对象方法的研究重点开始转向软件生命周期的分析阶段，并将 OOA 和 OOD 密切地联系在一起，出现了一大批面向对象的分析与设计（OOA&D）方法。截至 1994 年，公开发表并具有一定影响的 OOA&D 方法已达 50 余种。

由于各种 OOA 方法所强调的重点与该方法的主要特色不同，因此所产生的 OOA 模型从整体形态、结构框架到具体内容都有较大的差异。

1）OMT 方法简介

1991 年，James Rumbaugh 在《面向对象的建模与设计》（*Object-Oriented Modeling and Design*）一书中提出了面向对象分析与设计的 OMT（Object Modeling Technique）方法。20 世纪 90 年代中期，笔者曾使用 OMT 方法开发了“印典”、“书林”等排版系统。本书的 OOA 模型主要依据 OMT 方法，同时参考了 Peter Coad 和 Edward Yourdon 的 OOA 模型。

OMT 方法的 OOA 模型包括对象模型、动态模型和功能模型。

对象模型表示静态的、结构化的系统的“数据”性质。它是对模拟客观世界实体的对象及对象彼此间的关系的映射，描述了系统的静态结构。通常用类图表示。

动态模型表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。通常用状态图表示。

功能模型表示变化的系统的“功能”性质，它指明了系统应该“做什么”，因此更直接地反映了用户对目标系统的需求。通常用数据流图表示。

OMT 方法的三个模型，分别从三个不同侧面描述了所要开发的系统：功能模型指明了系统应该“做什么”；动态模型明确了什么时候做（即在何种状态下接受了什么事件的触发）；对象模型则定义了做事情的实体。这三种模型相互补充、相互配合，三者之间具有如下关系：

- 动态模型展示了对象模型中每个对象的状态及它接受事件和改变状态时所执行的操作；而功能模型中的处理则对应于对象模型中的对象所提供的服务。
- 对象模型展示了动态模型中是谁改变了状态和经受了操作；而功能模型中的处理则可能产生动态模型中的事件。
- 对象模型展示了功能模型中的动作者、数据存储和流的结构；而动态模型则展示了功能模型中执行加工的顺序。

2）建立对象模型

Peter Coad 和 Edward Yourdon 在 1991 年出版的《面向对象的分析》（*Object-Oriented Analysis*）一书中指出，复杂系统的对象模型通常由 5 个层次组成：类及对象层、结构层、主题层、属性层和服务层。上述 5 个层次对应着建立对象模型的 5 项主要活动：确定类与对象、确定结构与关联、划分主题、定义属性和定义服务。但这 5 项活动完全没必要顺序完成，也无须彻底完成一项活动之后再开始另外一项活动。

- 确定类与对象：类与对象是在问题域中客观存在的，系统分析的重要任务之一就是找出这些类与对象。首先找出所有候选的类与对象，然后进行反复筛选，删除不正确或不必要的类与对象。
- 确定结构与关联：结构与关联反映了对象（或类）之间的关系，主要有如下几种。
  - 一般－特殊结构（Generalization-Specialization Structure），又称为分类结构（Classification Structure），是由一组具有一般－特殊关系（继承关系）的类所组成的结构。一般－特殊关系（Generalization-Specialization Relation）的表达式为：is a kind of。

- ➢ 整体－部分结构（Whole-Part Structure），又称为组装结构（Composition Structure），是由一组具有整体－部分关系（组成关系）的类所组成的结构。整体－部分关系（Whole-Part Relation）的表达式为：has a。
- ➢ 实例关联（Instance Connection），即一个类的属性中含有另一个类的实例（对象），它反映了对象之间的静态联系。
- ➢ 消息关联（message connection），即一个对象在执行自己的服务时需要通过消息请求另一个对象为它完成某个服务，它反映了对象之间的动态联系。

- 划分主题：在开发大型、复杂软件系统的过程中，为了降低复杂程度，需要把系统划分成几个不同的主题。注意，应该按问题域而不是用功能分解方法来确定主题，应该按照使不同主题内的对象相互间依赖和交互最少的原则来确定主题。
- 定义属性：为了发现对象的属性，首先考虑借鉴以往的OOA结果，看看相同或相似的问题域是否有已开发的OOA模型，尽可能复用其中同类对象的属性定义。然后，按照问题域的实际情况，以系统责任为目标进行正确的抽象，从而找出每一对象应有的属性。
- 定义服务：发现和定义对象的服务，也应借鉴以往同类系统的OOA结果并尽可能加以复用。然后，研究问题域和系统责任以明确各个对象应该设立哪些服务，以及如何定义这些服务。

3）建立动态模型

建立动态模型的第一步，是编写典型交互行为的脚本。虽然脚本中不可能包括每个偶然事件，但至少必须保证不遗漏常见的交互行为。第二步，从脚本中提取出事件，确定触发每个事件的动作对象及接受事件的目标对象。第三步，排列事件发生的次序，确定每个对象可能有的状态及状态间的转换关系，并用状态图描绘它们。最后，比较各个对象的状态图，检查它们之间的一致性，确保事件之间的匹配。

4）建立功能模型

OMT 方法中的功能模型实际上就是结构化方法中的数据流图。从这点看，OMT 方法并不是“纯”面向对象的。这是 OMT 方法的一大缺陷。

1992 年，Ivar Jacobson 在《面向对象的软件工程——用例驱动的途径》（Object-Oriented Software Engineering，A Use Case Driven Approach）中首次提出了“用例”（Use Case）的概念。随后，有人提出以用例图取代数据流图进行需求分析和建立功能模型，这应该被看作对 OMT 方法的重大改进。使用用例图建立起来的系统模型也被称为用例模型。

一个用例是可以被行为者感受到的、系统的一个完整的功能。一幅用例图包含的模型元素有系统、行为者、用例及用例之间的关系。用例模型描述的是外部行为者所理解的系统功能。

目前，“用例驱动”已成为软件开发过程的一条重要原则。

3．面向对象的设计

1）OOA 与 OOD 的关系

与结构化方法不同，面向对象的方法并不强调分析与设计之间严格的阶段划分。OOA 与 OOD 所采用的概念、原则和表示法都是一致的，二者之间不存在鸿沟，不需要从分析文档到设计文档的转换，所以有些工作无论在分析时进行还是在设计时进行都不存在障碍。当然，OOA 与 OOD 仍然有不同的分工和侧重点。

关于 OOA 与 OOD 的关系，目前有两种不同的观点。

一种观点是继续沿用传统的分工——分析着眼于系统“做什么”，设计解决“怎么做”的问题。而 Peter Coad 和 Edward Yourdon 的 OOA&D 方法则采用了另外一种分工方式——分析阶段只考虑问题域和系统责任，建立一个独立于实现的 OOA 模型；设计阶段考虑与实现有关的因素，对 OOA 模型进行调整并补充与实现有关的部分，形成 OOD 模型。本书的 OOD 方法主要依据 Coad/Yourdon 的观点。

Coad/Yourdon 的 OOD 模型包括如下 4 个部件：人机交互部件、问题域部件、任务管理部件、数据管理部件。与此对应的 OOD 过程也包括 4 项活动，设计人机交互部件、设计问题域部件、设计任务管理部件和设计数据管理部件。

2）设计问题域部件

通过 OOA 所得出的问题域精确模型，为设计问题域部件奠定了良好的基础。通常，OOD 仅需从实现角度对问题域模型做一些补充和修改，主要是增添、合并或分解类与对象、属性及服务、调整继承关系等。

3）设计人机交互部件

在 OOA 过程中，已经对用户界面需求做了初步分析。在 OOD 过程中，则应该对系统的人机交互部件进行详细设计，以确定人机交互的细节，其中包括指定窗口和报表的形式、设计命令层次等内容。

4）设计任务管理部件

设计任务管理部件主要用于识别事件驱动任务，识别时钟驱动任务，识别优先任务，识别关键任务，识别协调任务，审查每个任务并定义每个任务。

5）设计数据管理部件

设计数据管理部件用于提供数据管理系统中存储和检索对象的基本结构，以及隔离具体的数据管理方案（如普通文件、关系数据库、面向对象数据库等）对其他部分的影响。

4.3 软件测试与软件维护

本节将介绍软件测试与软件维护。

4.3.1 软件测试

软件测试是软件质量保证的主要手段之一，也是在将软件交付给客户之前所必须完成的步骤。目前，软件的正确性证明尚未得到根本的解决，软件测试仍是发现软件错误和缺陷的主要手段。

大量统计资料表明，目前软件测试所花费用已超过软件开发费用的 30%。

1．软件测试基础

1）软件测试的目的

软件测试的目的就是在软件投入生产性运行之前，尽可能多地发现软件产品（主要是指程序）中的错误和缺陷。

为了发现程序中的错误，应竭力设计能暴露错误的测试用例。测试用例是由测试数据和预期结果构成的。一个好的测试用例是极有可能发现至今为止尚未发现的错误的测试用例。一次成功的测试是发现了至今为止尚未发现的错误的测试。

高效的测试是指用少量的测试用例，发现被测软件尽可能多的错误。

软件测试所追求的目标就是以尽可能少的时间和人力发现软件产品中尽可能多的错误。

2）软件测试准则

- 应该尽早地、不断地进行软件测试，把软件测试贯穿于开发过程的始终。
- 所有测试都应该能追溯到用户需求。从用户的角度看，最严重的错误是导致软件不能满足用户需求的那些错误。
- 应该从“小规模”测试开始，并逐步进行“大规模”测试。
- 应该远在测试之前就制定出测试计划。
- 根据Pareto原理，80%的错误可能出现在20%的程序模块中，测试成功的关键是怎样找出这20%的模块。
- 应该由独立的第三方从事测试工作。
- 对非法和非预期的输入数据也要像合法的和预期的输入数据一样编写测试用例。
- 检查软件是否做了应该做的事仅是成功的一半，另一半是看软件是否做了不该做的事。
- 在规划测试时不要设想程序中不会查出错误。
- 测试只能证明软件中有错误，不能证明软件中没有错误。

3）软件测试分类

- 从测试阶段划分，可分为单元测试、集成测试和确认测试。
- 从测试方法划分，可分为白盒测试、黑盒测试。

在实际应用中，一旦纠正了程序中的错误后，还应选择部分或全部原先已测试过的测试用例，对修改后的程序重新测试，这种测试称为回归测试。

2．单元测试

单元测试（Unit Testing），也称为模块测试，通常可放在编程阶段，由程序员对自己编写的模块自行测试，检查模块是否实现了详细设计说明书中规定的功能和算法。单元测试主要发现编程和详细设计中产生的错误，单元测试计划应该在详细设计阶段制定。

单元测试期间着重从如下几个方面对模块进行测试：模块接口、局部数据结构、重要的执行通路、出错处理通路、边界条件等。

测试一个模块时需要为该模块编写一个驱动模块和若干个桩（stub）模块。驱动模块用来调用被测模块，它接收测试者提供的测试数据，并把这些数据传送给被测模块，然后从被测模块接收测试结果，并以某种可以看见的方式（例如显示或打印）将测试结果返回给测试者。桩模块用来模拟被测模块所调用的子模块，它接受被测模块的调用，检验调用参数，并以尽可能简单的操作模拟被调用的子程序模块功能，把结果送回被测模块。顶层模块测试时不需要驱动模块，底层模块测试时不需要桩模块。

模块的内聚程度高可以简化单元测试过程。如果每个模块只完成一种功能，则需要的测试方案数目将明显减少，模块中的错误也更容易预测和发现。

3．集成测试

集成测试（Integration Testing），也称为组装测试，它是对由各模块组装而成的程序进行测试，主要目标是发现模块间的接口和通信问题。例如，数据穿过接口可能丢失；一个模块对另一个模块可能由于疏忽而造成有害影响；把子功能组合起来可能不产生预期的主

功能；个别看来是可以接受的误差可能积累到不能接受的程度；全程数据结构可能有问题等。集成测试主要发现设计阶段产生的错误，集成测试计划应该在概要设计阶段制定。

集成的方式可分为非渐增式和渐增式。

非渐增式集成是先测试所有的模块，然后一下子把所有这些模块集成到一起，并把庞大的程序作为一个整体来测试。这种测试方法的出发点是可以“一步到位”，但测试者面对众多的错误现象，往往难以分清哪些是“真正的”错误，哪些是由其他错误引起的“假性错误”，诊断定位和改正错误也十分困难。非渐增式集成只适合一些非常小的软件。

渐增式集成是将单元测试和集成测试合并到一起，它根据模块结构图，按某种次序选一个尚未测试的模块，把它同已经测试好的模块组合在一起进行测试，每次增加一个模块，直到所有模块被集成在程序中。这种测试方法比较容易定位和改正错误，目前在进行集成测试时已普遍采用渐增式集成。

渐增式集成又可分为自顶向下集成和自底向上集成。自顶向下集成先测试上层模块，再测试下层模块。由于测试下层模块时它的上层模块已测试过，所以不必另外编写驱动模块。自底向上集成先测试下层模块，再测试上层模块。同样，由于测试上层模块时它的下层模块已测试过，所以不必另外编写桩模块。这两种集成方法各有利弊，一种方法的优点恰好对应于另一种方法的缺点，实际测试时可根据软件特点及进度安排灵活选用最适当的方法，也可将两种方法混合使用。

4．确认测试

确认测试（Validation Testing）主要依据软件需求说明书检查软件的功能、性能及其他特征是否与用户的需求一致。确认测试计划应该在需求分析阶段制定。

软件配置复查是确认测试的另一项重要内容。复查的目的是保证软件配置的所有成分都已齐全，质量符合要求，文档与程序完全一致，具有完成软件维护所必需的细节。

如果一个软件是为某个客户定制的，最后还要由该客户来实施验收测试（Acceptance Testing），以便确认其所有需求是否都已得到满足。由于软件系统的复杂性，在实际工作中，验收测试可能会持续到用户实际使用该软件之后的相当长的一段时间。

如果一个软件是作为产品被许多客户使用的，不可能也没必要由每个客户进行验收测试。绝大多数软件开发商都使用被称为α（Alpha）测试和β（Beta）测试的过程，来发现那些看起来只有最终用户才能发现的错误。

α测试由用户在开发者的场所进行，并且在开发者的指导下进行测试。开发者负责记录发现的错误和使用中遇到的问题。也就是说，α测试是在“受控的”环境中进行的。

β测试是在一个或多个用户的现场由该软件的最终用户实施的，开发者通常不在现场，用户负责记录发现的错误和使用中遇到的问题并把这些问题报告给开发者。也就是说，β测试是在“非受控的”环境中进行的。

经过确认测试之后的软件即可交付使用。

5．白盒测试

白盒测试，又称为结构测试，主要用于单元测试阶段。它的前提是把程序看成装在一个透明的白盒子里，测试者完全知道程序的结构和处理算法。这种方法按照程序内部逻辑设计测试用例，检测程序中的主要执行通路是否都能按预定要求正确工作。

白盒测试常用的技术是逻辑覆盖，即考查用测试数据运行被测程序时对程序逻辑的覆盖程度。主要的覆盖标准有 6 种：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、组合条件覆盖和路径覆盖。

1）语句覆盖

语句覆盖是指选择足够多的测试用例，使得运行这些测试用例时，被测程序的每个语句至少执行一次。

很显然，语句覆盖是一种很弱的覆盖标准。考虑如图 4-7 所示的源程序流程图。

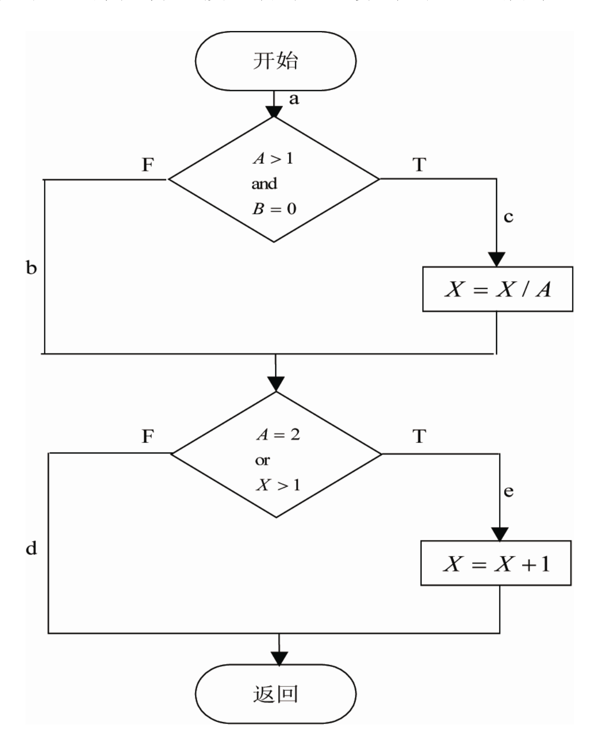


图 4-7　覆盖用例设计的源程序流程图

假设事先选取测试路径如下：

$L1(a \to c \to e)$

$= \{(A > 1)\text{and}(B = 0)\}\text{and}\{(a = 2)\text{or}(X / A > 1)\}$

$= \{(A > 1)\text{and}(B = 0)\}\textit{and}\{(A = 2)\text{or}\{(A > 1)\text{and}\{(B = 0)\text{and}(X / A > 1)\}$

$= \{(A = 2)\text{and}(B = 0)\}\text{or}\{(A > 1)\text{and}(B = 0)\text{and}(X / A > 1)\}$

$L2(a \to b \to d)$

$= \text{not}\{(A > 1)\text{and}(B = 0)\}\text{ and not}\{(a = 2)\text{or}(X / A > 1)\}$

$= \{\text{not}(A > 1)\text{ or not}(B = 0)\}\text{and}\{\text{not}(A = 2)\text{ and not}(X > 1)\}$

$= \{\text{not}(A > 1)\text{ and not}(A = 2)\}\text{and}\{\text{not}(X > 1)\text{ or not}(B = 0)\}$

$\text{and}\{\text{not}(A = 2)\text{ and not}(X > 1)\}$

$L3(a \to b \to e)$

$= \text{not}\{(A > 1)\text{and}(B = 0)\}\text{and}\{(A = 2)\text{or}(X > 1)\}$

$= \{\text{not}(A > 1)\text{or not}(B = 0)\}\text{and}\{(A = 2)\text{or}(A > 1)\}$

$= \{\text{not}(A > 1)\text{and}(A = 2)\}\text{or}\{\text{not}(A > 1)\text{and}(X > 1)\}$

$\text{or}\{\text{not}(B = 0)\text{and}(A = 2)\}\text{or}\{\text{not}(B = 0)\text{and}(X > 1)\}$

$L4(a \to c \to d)$
$= \{(A>1)\text{and}(B=0)\}\text{and not}\{(A=2)\text{or}(X/A>1)\}$
$= \{(A>1)\text{and}(B=0)\}\text{and}\{\text{not}(A=2)\text{and not}(X/A>1)\}$

假设测试用例的设计格式如下：

输入的是[A,B,X]，输出的是[A,B,X]。

为图 4-7 设计的满足语句覆盖的测试用例是：

[2，0，4]，[2，0，3]

该用例可以覆盖路径：

$$L1(a \to c \to e): \begin{array}{l}\{(A=2)\text{and}(B=0)\}\text{or}\\ \{(A>1)\text{and}(B=0)\text{and}(X/A>1)\}\end{array}$$

至此，所有的可执行语句包括两个判断语句、两个赋值语句，均已被执行。

语句覆盖度量的主要好处是它可以直接应用在目标码上，不需要对源代码进行处理。执行轮廓就完成了这个度量。语句覆盖的主要缺点是对一些控制结构很迟钝。例如，考虑如下 C/C++代码：

```
int *p=NULL;
if (condition)
    p=&variable;
*p=123;
```

如果在 condition 取假的情况下，语句覆盖率显示这 3 句都覆盖到了，但是代码执行是失败的。这是语句覆盖率的严重的缺陷，IF 语句是很普通的一种情况。另外语句覆盖不能报告循环是否到达它们的终止条件——只能显示循环是否被执行。do-while 循环通常要至少执行一次，语句覆盖认为它们和无分支语句是一样的。语句覆盖对逻辑运算符反映是迟钝的（|| and &&）。语句覆盖不能区分连续的 switch 语句。

2）判定覆盖

判定覆盖又称为分支覆盖，它的含义是，不仅每个语句至少执行一次，而且每个判定的每种可能的结果（分支）都至少执行一次。判定覆盖比语句覆盖强，但对程序逻辑的覆盖程度仍然不高。

对于图 4-7 中的程序流程图，选择如下路径 $L1(a \to c \to e)$、$L2(a \to b \to d)$，就可以得到满足判定覆盖要求的测试用例：

[(2,0,4),(2,0,3)]、[(1,1,1),(1,1,1)]

测试用例中[(2,0,4),(2,0,3)]可以覆盖路径：

$L1(a \to c \to e)$
$= \{(A=2)\text{and}(B=0)\}\text{or}\{(A>1)\text{and}(B=0)\text{and}(X/A>1)\}$

[(1,1,1),(1,1,1)]可以覆盖路径：

$L2(a \to b \to d)$
$= \{\text{not}(A>1)\text{and not}(A=2)\}\text{and}\{\text{not}(X>1)$
$\text{or not}(B=0)\}\text{and}\{\text{not}(A=2)\text{and not}(X>1)\}$

另外选择路径 $L3(a \to b \to e)$、$L4(a \to c \to d)$，就可以得到满足判定覆盖要求的测试用例：

[(2,1,1),(2,1,2)]、[(3,0,3),(3,1,1)]

[(2,1,1),(2,1,2)]可以覆盖：

$$L3(a \to b \to e)$$
$$= \{\text{not}(A > 1)\,\text{and}(X > 1)\}$$
$$\text{or}\{\text{not}(B = 0)\,\text{and}(A = 2)\}\,\text{or}\{\text{not}(B = 0)\,\text{and}(X > 1)\}$$

[(3,0,3),(3,1,1)]可以覆盖：

$$L4(a \to c \to d)$$
$$= \{(A > 1)\,\text{and}(B = 0)\}\,\text{and}\{\text{not}(A = 2)\,\text{and not}(X / A > 1)\}$$

判定覆盖报告是否为布尔型的表达式取值 true 和 false 在控制结构中被测试到了，整个布尔型的表达式被认为是一个整体，而不考虑内部是否包含逻辑与（and）或逻辑或（or）操作符。另外包括 switch-statement、exception handlers 和 interrupt handlers 的覆盖。

判定覆盖具有语句覆盖的简单性，但是没有语句覆盖的问题。缺点是这个度量忽略了在布尔型表达式内部的布尔取值。比如考虑如下的 C/C++/Java 代码：

```
if(condition1 && (condition2 || function1()))
    statement1;
    else
statement2;
```

这个判断条件可以完全不用调用 function1。测试表达是真时可以取 condition1 为 true 和 condition2 为 true，测试表达为假时可以取 condition1 为 false。

3）条件覆盖

条件覆盖的含义是，不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。条件覆盖不一定包含判定覆盖，判定覆盖也不一定包含条件覆盖。

在设计条件覆盖测试用例时，可以先对所有条件的取值加以标记。例如：

对于图中的第一个判断，条件 $A > 1$ 时取真为 T_1，取假为 $\overline{T_1}$；条件 $B = 0$ 时取真为 T_2，取假为 $\overline{T_2}$。

对于图中的第二个判断，条件 $A = 2$ 时取真为 T_3，取假为 $\overline{T_3}$；条件 $X > 1$ 时取真为 T_4，取假为 $\overline{T_4}$。

可以选取测试用例如下，如表 4-12 所示。

表 4-12　条件覆盖测试用例 1

测试用例	覆盖分支	条件取值
[(2,0,4),(2,0,3)]	$L1(a \to c \to e)$ $= \{(A = 2)\,\text{and}(B = 0)\}\,\text{or}\{(A > 1)\,\text{and}(B = 0)\,\text{and}(X / A > 1)\}$	$T_1 T_2 T_3 T_4$
[(1,0,1),(1,0,1)]	$L2(a \to b \to d)$ $= \{\text{not}(A > 1)\,\text{or not}(B = 0)\}\,\text{and}\{\text{not}(A = 2)\,\text{and not}(X > 1)\}$	$\overline{T_1} T_2 \overline{T_3}\,\overline{T_4}$
[(2,1,1),(2,1,2)]	$L3(a \to b \to e)$ $= \text{not}\{(A > 1)\,\text{and}(B = 0)\}\,\text{and}\{(A = 2)\,\text{or}(X > 1)\}$	$T_1 \overline{T_2} T_3 \overline{T_4}$

也可以选取测试用例如下，如表 4-13 所示。

表 4-13　条件覆盖测试用例 2

测 试 用 例	覆 盖 分 支	条 件 取 值
[(1,0,3),(1,0,4)]	$L3(a \to b \to e)$ $= \text{not}\{(A>1)\text{and}(B=0)\}\text{and}\{(A=2)\text{or}(X>1)\}$	$\overline{T_1}T_2\overline{T_3}T_4$
[(2,1,1),(2,1,2)]	$L3(a \to b \to e)$ $= \{\text{not}(A>1)\text{and}(A=2)\}\text{or}\{\text{not}(A>1)\text{and}(X>1)\}$ $\text{or}\{\text{not}(B=0)\text{and}(A=2)\}\text{or}\{\text{not}(B=0)\text{and}(X>1)\}$	$T_1\overline{T_2}T_3\overline{T_4}$

完全的条件覆盖并不能保证完全的判定覆盖。例如，考虑如下的 C++/Java 代码。

```
Bool f(bool e) {return false;}
Bool a[2]={false,false};
If(f(a &&b))…
If(a[int(a && b)])…
If((a && b)?false :false)…
```

所有 3 个 if 语句不管 a 和 b 取值是什么，判定覆盖率只能达到 50%，但是条件覆盖率却能达到 100%。

4）判定/条件覆盖

同时满足判定覆盖和条件覆盖的逻辑覆盖称为判定/条件覆盖。它的含义是，选取足够的测试用例，使得判定表达式中每个条件的所有可能结果至少出现一次，而且每个判定本身的所有可能结果也至少出现一次。

5）条件组合覆盖

条件组合覆盖的含义是，选取足够的测试用例，使得每个判定表达式中条件结果的所有可能组合至少出现一次。

显然，满足条件组合覆盖的测试用例，也一定满足判定/条件覆盖。因此，条件组合覆盖是上述 5 种覆盖标准中最强的一种。然而，条件组合覆盖还不能保证程序中所有可能的路径都至少经过一次。

6）路径覆盖

路径覆盖的含义是，选取足够的测试用例，使得程序的每条可能执行到的路径都至少经过一次（如果程序中有环路，则要求每条环路径至少经过一次）。

路径覆盖实际上考虑了程序中各种判定结果的所有可能组合，因此是一种较强的覆盖标准。但路径覆盖并未考虑判定中的条件结果的组合，并不能代替条件覆盖和条件组合覆盖。

为图 4-7 所示的程序代码段设计的测试用例如表 4-14 所示。

表 4-14　路径覆盖测试用例

测 试 用 例	通 过 路 径	覆 盖 条 件
[(2,0,4),(2,0,3)]	$L1(a \to c \to e)$ $= \{(A=2)\ \text{and}(B=0)\}\ \text{or}$ $\{(A>1)\ \text{and}(B=0)\ \text{and}(X/A>1)\}$	$T_1T_2T_3T_4$
[(1,1,1),(1,1,1)]	$L2(a \to b \to d)$ $= \{\text{not}(A>1)\ \text{or}\ \text{not}(B=0)\}$ $\text{and}\{\text{not}(A=2)\ \text{and}\ \text{not}(X>1)\}$	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{T_4}$
[(1,1,2),(1,1,3)]	$L3(a \to b \to e)$ $= \text{not}\{(A>1)\ \text{and}(B=0)\}\ \text{and}\{(A=2)\ \text{or}(X>1)\}$	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}T_4$
[(3,0,3),(3,0,1)]	$L4(a \to c \to d)$ $= \{(A>1)\ \text{and}(B=0)\}\ \text{and}\{\text{not}(A=2)\ \text{and}\ \text{not}(X/A>1)\}$	$T_1T_2\overline{T_3}\,\overline{T_4}$

路径覆盖的好处是可以对程序段进行彻底的测试，但有如下两个缺点。

一是路径是以分支的指数级别增加的，比如，一个函数包含 10 个 IF 语句，就有 2^{10}=1 024 个路径要测试。如果再多加一个 IF 语句，路径数就达到 2 048 个。

二是许多路径不可能与执行的数据无关。例如：

```
if(success)
    statement1;
statement2;
if(success)
    statement3;
```

路径覆盖认为上述语句包含 4 个路径，实际上只有两个是可行的：success=false 和 success=true。

6．黑盒测试

黑盒测试，又称为功能测试，主要用于集成测试和确认测试阶段。它把软件看作一个不透明的黑盒子，完全不考虑（或不了解）软件的内部结构和处理算法，它只检查软件功能是否能按照软件需求说明书的要求正常使用，软件是否能适当地接收输入数据并产生正确的输出信息，软件运行过程中能否保持外部信息（例如文件和数据库）的完整性等。

常用的黑盒测试技术包括等价类划分、边值分析、错误推测和因果图等。

1）等价类划分

在设计测试用例时，等价类划分是用得最多的一种黑盒测试方法。所谓等价类就是某个输入域的集合，对于一个等价类中的输入值来说，它们揭示程序中错误的作用是等效的。也就是说，如果等价类中的一个输入数据能检测出一个错误，那么等价类中的其他输入数据也能检测出同一个错误；反之，如果等价类中的一个输入数据不能检测出某个错误，那么等价类中的其他输入数据也不能检测出这一错误（除非这个等价类的某个子集还属于另一等价类）。

如果一个等价类内的数据是符合（软件需求说明书）要求的、合理的数据，则称这个等价类为有效等价类。有效等价类主要用来检验软件是否实现了软件需求说明书中规定的功能。

如果一个等价类内的数据是不符合（软件需求说明书）要求的、不合理或非法的数据，则称这个等价类为无效等价类。无效等价类主要用来检验软件的容错性。

黑盒测试中，利用等价类划分方法设计测试用例的步骤如下。

- 根据软件的功能说明，对每一个输入条件确定若干个有效等价类和若干个无效等价类，并为每个有效等价类和无效等价类编号。
- 设计一个测试用例，使其覆盖尽可能多的尚未被覆盖的有效等价类。重复这一步，直至所有的有效等价类均被覆盖。
- 设计一个测试用例，使其覆盖一个尚未被覆盖的无效等价类。重复这一步，直至所有的无效等价类均被覆盖。

应当特别注意，无效等价类用来测试非正常的输入数据，因此每个无效等价类都有可能查出软件中的错误，所以要为每个无效等价类设计一个测试用例。

下面以一个经典的三角形问题为例说明采用等价类划分方法的测试用例设计。

问题描述：三角形问题接受 3 个整数 a、b 和 c 作为输入，用作三角形的边。整数 a、b 和 c 必须满足如下条件。

c_1.1≤a≤200　　c_2.1≤b≤200　　c_3.1≤c≤200

c_4.a＜b+c　　c_5.b＜a+c　　c_6.c＜a+b

程序的输出是由这三条边确定的三角形类型：等边三角形、等腰三角形、不等边三角形或非三角形。如果输入值没有满足这些条件中的任何一个，则程序会通过输出消息来进行通知，例如，“b 的取值不在容许的取值范围内”。如果取值 a、b 和 c 满足 c_1、c_2 和 c_3，则给出如下 4 种相互排斥输出中的一个：

- 如果三条边相等，则程序的输出是等边三角形。
- 如果恰好有两条边相等，则程序的输出是等腰三角形。
- 如果没有两条边相等，则程序输出的是不等边三角形。
- 如果 c_4、c_5 和 c_6 中有一个条件不满足，则程序输出的是非三角形。

等价类划分的测试用例如表 4-15 所示。

表 4-15　等价类划分的测试用例

测 试 用 例	a	b	c	预 期 输 出
（1）	5	5	5	等边三角形
（2）	2	2	1	等腰三角形
（3）	3	4	5	不等边三角形
（4）	4	1	2	非三角形
（5）	−1	5	5	a 取值越界
（6）	5	−1	5	b 取值越界
（7）	5	5	−1	c 取值越界
（8）	201	5	5	a 取值越界
（9）	5	201	5	b 取值越界
（10）	5	5	201	c 取值越界
（11）	−1	−1	5	a、b 取值越界
（12）	5	−1	−1	b、c 取值越界
（13）	−1	5	−1	a、c 取值越界
（14）	−1	−1	−1	a、b、c 取值越界

2）边值分析

经验表明，软件在处理边界情况时最容易出错。设计一些测试用例，使软件恰好运行在边界附近，暴露出软件错误的可能性会更大一些。

通常，每一个等价类的边界，都应该着重测试，选取的测试数据应该恰好等于、稍小于或稍大于边界值。

将等价类划分法和边值分析法结合使用，更有可能发现软件中的错误。

3）错误推测

错误推测使用等价类划分和边值分析技术，有助于设计出具有代表性的、容易暴露软件错误的测试方案。但是，不同类型、不同特征的软件通常又有一些特殊的容易出错的地方。错误推测法主要依靠测试人员的经验和直觉，从各种可能的测试方案中选出一些最可能引起程序出错的方案。

4）因果图

因果图法是根据输入条件与输出结果之间的因果关系来设计测试用例的，它首先检查输入条件的各种组合情况，并找出输出结果对输入条件的依赖关系，然后为每种输出条件的组合设计测试用例。

4.3.2 软件维护

软件维护是指在软件交付使用之后直至软件被淘汰的整个时期内为了改正错误或满足新的需求而修改软件的活动。

软件维护的代价是很大的，据 1994 年 Software Engineering Encyclopedia 记载，20 世纪 80 年代末用于软件维护的花费约为整个软件生命周期总花费的 75%，而且还在逐年上升。

1．软件维护类型

根据引起软件维护的原因，软件维护通常可分为如下 4 种类型：

1）改正性维护

改正性维护是指在使用过程中发现了隐蔽的错误后，为了诊断和改正这些隐蔽错误而修改软件的活动。

2）适应性维护

适应性维护是指为了适应变化了的环境而修改软件的活动。

3）完善性维护

完善性维护是指为了扩充或完善原有软件的功能或性能而修改软件的活动。

4）预防性维护

预防性维护是指为了提高软件的可维护性和可靠性、为未来的进一步改进打下基础而修改软件的活动。

2．软件的可维护性

软件的可维护性是指理解、改正、改动、改进软件的难易程度。根据 Boehm 质量模型，通常影响软件可维护性的因素有可理解性、可测试性和可修改性。

1）可理解性

可理解性是指维护人员理解软件的结构、接口、功能和内部过程的难易程度。

2）可测试性

可测试性是指测试和诊断软件错误的难易程度。

3）可修改性

可修改性是指修改软件的难易程度。

为了提高软件的可维护性，在软件生命周期的各个阶段都必须充分考虑维护问题。先进的软件工程方法是软件可维护的基础保证。

面向对象方法学的对象封闭机制、消息通信机制、继承机制和多态机制从根本上提高了软件的可理解性、可测试性和可修改性。

结构化设计的几条主要原则，如模块化、信息隐蔽、高内聚、低耦合等，对于提高软件的可理解性、可测试性和可修改性也都有重要的作用。

另外，书写详细正确的文档、书写源文件的内部注解、使用良好的编程语言、具有良好的程序设计风格，也有助于提高软件的可理解性。使用先进的测试工具、保存以前的测试过程和测试用例，则有助于提高软件的可测试性。

3．软件维护管理

软件维护管理是指为保证维护质量、提高维护效率、控制维护成本而进行的维护过程管理，它要求对软件的每次“修改”均需经过申请、评估、批准、实施、验证等步骤。

软件维护管理的核心是维护评估和维护验证。维护评估的主要工作包括：判定维护申请的合理性与轻重缓急、确定维护的可行性与时间及费用、制定维护策略与维护计划等。维护验证主要审查修改后的软件是否实现了维护目标、软件文档是否也做了相应修改等。

4.4　软件工具与软件开发环境

本节将介绍软件工具与软件开发环境。

4.4.1　软件工具

软件工具是指用于辅助软件开发、运行、维护、管理、支持等过程中的活动的软件，通常也称为计算机辅助软件工程（Computer Aided Software Engineering，CASE）工具。

软件开发工具种类繁多，很难有一种统一分类方法。由于大多数软件工具仅限于支持软件生命周期过程中的某些特定的活动，通常可按软件过程的活动分为软件开发工具、软件维护工具和软件管理工具等。

1．软件开发工具

1）需求分析工具

需求分析工具主要包括支持结构化方法的数据流图、数据字典和支持面向对象方法的类图、用例图和状态图等。

2）设计工具

设计工具主要包括概要设计阶段的模块结构图、层次图、HIPO 图和详细设计阶段的程序流程图、盒图（N－S 图）、PAD 图和过程设计语言（PDL）等。

面向对象的设计工具一般与分析阶段使用的工具一致，可以通称为分析设计工具。

3）编程工具

编程工具主要包括编辑程序、汇编程序、编译程序、构造程序（Builder）和调试程序等。

4）测试工具

测试工具包括静态分析工具、动态测试工具和测试数据自动生成工具等。

2．软件维护工具

1）版本控制工具

版本控制工具用来存储、更新、恢复和管理一个软件的多个版本。

2）文档分析工具

文档分析工具用来对软件开发过程中形成的文档进行分析，给出软件维护活动所需的维护信息。

3）开发信息库工具

开发信息库工具用来维护软件项目的开发信息，包括对象、模块等。

4）逆向工程工具

逆向工程工具在软件生命周期中，将某种形式表示的软件转换成更高抽象形式表示的软件活动称为逆向工程。逆向工程工具就是辅助软件人员进行这种逆向工程活动的软件工具，如反汇编工具、反编译工具等。

5）再工程工具

再工程工具用来支持重构一个功能和性能更为完善的软件系统。目前的再工程工具主要集中在代码重构、程序结构重构和数据重构等方面。

3．软件管理工具

1）项目管理工具

项目管理工具用来辅助软件的项目管理活动（包括项目的计划、调度、通信、成本估算、资源分配及质量控制等）。

2）配置管理工具

配置管理工具用来辅助完成软件配置项的标识、版本控制、变化控制、审计和状态统计等基本任务，使各配置项的存取、修改和系统生成易于实现，从而简化审计过程、改进状态统计、减少错误、提高系统质量。

3）软件评价工具

软件评价工具用来辅助管理人员进行软件质量保证的有关活动。

4.4.2 软件开发环境

软件开发环境是指支持软件产品开发的软件系统。

集成型软件开发环境是一种把支持多种软件开发方法和开发模型、支持软件开发全过程的软件工具集成在一起的软件开发环境。这种环境通常应具有开放性和可剪裁性。开放性为将环境外的工具集成到环境中来提供方便；可剪裁性根据不同的应用或不同的用户需求进行剪裁，以形成特定的开发环境。

集成型开发环境通常可由工具集成和环境集成机制两部分组成。环境集成机制主要有数据集成机制、控制集成机制和界面集成机制。

1）数据集成机制

数据集成机制提供统一的数据模式和数据接口规范，需要相互协作的工具通过这种统一的模式与规范交换数据。数据集成可以有不同的层次，如共享文件、共享数据结构和共享信息库等。

2）控制集成机制

控制集成机制支持各工具或各开发活动之间的通信、切换、调度和协同工作，并支持软件开发过程的描述、执行和转接。通常使用消息通信机制实现控制集成，工具间发送的消息统一由消息服务器进行管理。

3）界面集成机制

界面集成机制为统一的工具界面风格和统一的操作方式提供支持，使得环境中的工具具有相同的视觉效果和操作规则，减少用户为学习不同工具的使用所花费的开销。界面集成主要体现在相同或相似的窗口、菜单、工具条、快捷键、操作规则与命令语法等。

4.5 软件质量保证

本节将介绍软件质量保证。

4.5.1 软件质量

概括地说，软件质量就是软件与明确地和隐含地定义的需求相一致的程度。具体地说，软件质量是软件与明确叙述的功能和性能需求、文档中明确描述的开发标准，以及任何专业开发的软件产品都应该具有的隐含特征相一致的程度。

软件质量具有如下 3 个要点：

- 用户需求是衡量软件质量的基础，与需求不一致就无质量可言。
- 指定的开发标准定义了一组指导软件开发的准则。如果没有遵守这些准则，肯定会导致软件质量不高。
- 通常还有一些没有明确写进用户需求说明书但开发人员都应当了解的隐含需求（例如易理解性、易修改性等）。如果软件仅满足明确描述的需求，但不满足这些隐含的需求，那么软件的质量仍然是值得怀疑的。

计算机软件是一种复杂、抽象的逻辑实体，它所固有的一些特点包括：抽象性、复杂性、多样性、易变性、软件开发需求难于把握等。所有这些软件独具的特点都增加了软件开发的困难。

影响软件质量的因素主要包括：

- 人的因素。
- 软件需求。
- 质量问题可能出现在开发过程的各个环节上。
- 测试的局限性。
- 质量管理的困难。

- 质量管理未能给予足够的重视。
- 软件人员的传统习惯。
- 开发规范。
- 开发工具的支持不够。

4.5.2 软件质量特性

软件质量特性可用多种软件质量模型来描述。本书只介绍 ISO/IEC 9126 软件质量模型和 Mc Call 软件质量模型。

1. ISO/IEC 9126 软件质量模型

国际标准化组织和国际电工委员会发布了关于软件质量的标准 ISO/IEC 9126－1991。中国于 1996 年将其等同采用，成为国家标准《GB/T16260－1996 软件产品评价、质量特性及其使用指南》。ISO/IEC 9126 软件质量模型由 3 个层次组成：第一层是 6 个质量特性，第二层是 21 个质量子特性，第三层是度量指标。该模型的质量特性和质量子特性如表 4-16 所示。

表 4-16 ISO/IEC 9126 软件质量模型的质量特性和质量子特性

质 量 特 性	质量子特性
功能性（Functionality）	适宜性（Suitability）
	准确性（Accurateness）
	互用性（Interoperability）
	依从性（Compliance）
	安全性（Security）
可靠性（Reliability）	成熟性（Maturity）
	容错性（Fault Tolerance）
	可恢复性（Recoverability）
可用性（Usability）	可理解性（Understandability）
	易学性（Learnability）
	可操作性（Operability）
效率（Efficiency）	时间特性（Time Behavior）
	资源特性（Resource Behavior）
可维护性（Maintainability）	可分析性（Analyzability）
	可修改性（Changeability）
	稳定性（Stability）
	可测试性（Testability）
可移植性（Portability）	适应性（Adaptability）
	易安装性（Installability）
	一致性（Conformance）
	可替换性（Replaceability）

1）功能性

功能性是指与功能及其指定的性质有关的一组软件属性。

- 适宜性：规定任务提供一组功能的能力及这组功能的适宜程度。
- 准确性：系统满足需求规格说明和用户目标的程度，即在预定环境下能正确地完成预期功能的程度。
- 互用性：同其他指定系统的协同工作能力。
- 依从性：软件服从有关标准、约定、法规及类似规定的程度。
- 安全性：避免对程序及数据的非授权故意或意外访问的能力。

2）可靠性

可靠性是指与软件在规定的一段时间内和规定的条件下维持其性能水平有关的一组软件属性。

- 成熟性：由软件故障引起失效的频度。
- 容错性：在软件错误或违反指定接口的情况下维持指定性能水平的能力。
- 可恢复性：在故障发生后重新建立其性能水平、恢复直接受影响数据的能力，以及为达此目的所需的时间与工作量。

3）可用性

可用性是指与使用的难易程度及规定或隐含用户对使用方式所做的评价有关的软件属性。

- 可理解性：用户理解该软件系统的难易程度。
- 易学性：用户学习使用该软件系统的难易程度。
- 可操作性：用户操作该软件系统的难易程度。

4）效率

效率是指与在规定条件下软件的性能水平与所用资源量之间的关系有关的一组软件属性。

- 时间特性：响应和处理时间及软件执行其功能时的吞吐量。
- 资源特性：软件执行其功能时，所使用的资源量及使用资源的持续时间。

5）可维护性

可维护性是指与软件维护的难易程度有关的一组软件属性。

- 可分析性：诊断缺陷或失效原因、判定待修改程序的难易程度。
- 可修改性：修改、排错或适应环境变化的难易程度。
- 稳定性：修改造成难以预料的后果的风险程度。
- 可测试性：测试已修改软件的难易程度。

注意：ISO/IEC 9126 中“可维护性”特性所包含的“子特性”与“软件的可维护性”一节中介绍的 Boehm 质量模型中影响可维护性的因素在表达上略有不同，主要增加了“稳定性”这一项。

6）可移植性

可移植性是指与软件可从某一环境转移到另一环境的能力有关的一组软件属性。

- 适应性：软件无须采用特殊处理就能适应不同的规定环境的程度。
- 易安装性：在指定环境下安装软件的难易程度。

- 一致性：软件服从与可移植性有关的标准或约定的程度。
- 可替换性：软件在特定软件环境中用来替代指定的其他软件的可能性和难易程度。

2．Mc Call 软件质量模型

Mc Call 软件质量模型从软件产品的运行、修正、转移三个方面确定了 11 项软件质量特性。本书介绍的是增加了“健壮性”、“风险性”和“可理解性”的扩充 Mc Call 模型，如表 4-17 所示。

表 4-17 Mc Call 模型

产 品 活 动	质 量 特 性	描 述
产品运行（Product Operation）	正确性（Correctness）	它按我的需要工作吗？
	健壮性（Roborance）	它能应付意外事件吗？
	效率（Efficiency）	它需要的资源多吗？
	完整性（Integrity）	它是安全的吗？
	可用性（Usability）	我能使用它吗？
	风险性（Venture）	能按预定计划完成它吗？
	可靠性（Reliability）	它是可靠的吗？
产品修正（Product Revision）	可理解性（Understandability）	我能理解它吗？
	可维护性（Maintainability）	我能修复它吗？
	灵活性（Flexibility）	我能改变它吗？
	可测试性（Testablity）	我能测试它吗？
产品转移（Product Transition）	可移植性（Portability）	我能在另外的环境使用它吗？
	可复用性（Reusablity）	我能复用它的某些部分吗？
	互运行性（Interoperability）	我能把它和另一系统结合吗？

Mc Call 模型中质量特性的准确定义与 ISO/IEC 9126 模型中对应项的定义大同小异，在此不再重复定义。应当注意：Mc Call 模型中的“完整性”相当于 ISO/IEC 9126 模型中的“安全性”，而 Mc Call 模型中的“健壮性”虽与 ISO/IEC 9126 模型中的“容错性”有点相似但并不完全相同。健壮性的定义是：在硬件发生故障、输入的数据无效或操作错误等意外环境下，系统能做出适当响应的程度。

4.5.3 软件质量保证

软件质量保证是为保证软件系统充分满足用户要求的质量而进行的有计划、有组织的活动，其目的是生产高质量的软件。

1．软件质量保证的困难与主要手段

软件质量保证的主要困难表现在如下几个方面。

- 软件开发的管理人员往往更关心项目开发的成本与进度。因为成本和进度是显而易见的，而软件质量则难以度量。
- 如果软件开发的管理人员对于交付的软件含有多少隐患不必负任何责任，则他们必定没有太高的热情去控制开发的质量，更不必说保证质量。
- 开发人员的习惯一旦形成便难以改变，他们的行为也难于控制。而高质量的软件产品，又主要取决于参与开发的人员。

- 复杂的软件项目需要许多技术人员和管理人员参与，对问题的不同认识和误解如不能及时消除，必然影响软件质量。
- 软件开发人员的频繁流动，特别是骨干开发人员的流失，也会使软件质量受到一定影响。

软件质量保证的主要手段如下。

- 开发初期制订质量保证计划，并在开发中坚持实行。
- 开发前选定或制订开发标准或开发规范，并遵照实施。
- 从选择分析设计方法和工具，形成高质量的分析模型和设计模型。
- 严格执行阶段评审，以便及时发现问题。
- 各个开发阶段的测试。
- 对软件的每次“变动”都要经过申请、评估、批准、实施、验证等步骤。
- 软件质量特性的度量化。
- 软件生存期的各阶段都要有完整的文档。

2．CMM

CMM 是软件过程能力成熟度模型（Capacity Maturity Model）的简称，是美国卡耐基梅隆大学软件工程研究所（CMU/SEI）为了满足美国联邦政府评估软件供应商能力的要求，于 1986 年开始研究的模型，并于 1991 年正式推出了 CMM 1.0 版。CMM 自问世以来备受关注，在一些发达国家和地区得到了广泛应用，成为衡量软件企业软件开发和管理水平的重要参考因素，以及软件过程改进事实上的工业标准。

CMM 模型描述和分析了软件过程能力的发展程度，确立了一个软件过程成熟程度的分级标准，如图 4-8 所示。

- 初始级：软件过程的特点是无秩序的，有时甚至是混乱的。软件过程定义几乎处于无章法和步骤可循的状态，软件产品所取得的成功往往依赖极个别人的努力和机遇。初始级的软件过程是未加定义的随意过程，项目的执行是随意甚至是混乱的。也许，有些企业制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，且执行没有政策、资源等方面的保证时，那么它仍然被视为初始级。

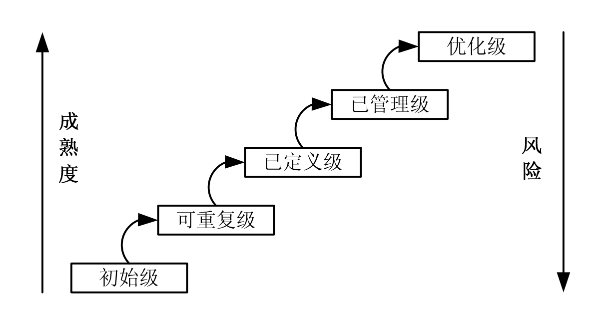


图 4-8　软件过程成熟度的级别

- 可重复级：已经建立了基本的项目管理过程，可用于对成本、进度和功能特性进行跟踪。对类似的应用项目，有章可循并能重复以往所取得的成功。焦点集中在软件管理过程上。一个可管理的过程则是一个可重复的过程，一个可重复的过程

则能逐渐演化和成熟。从管理角度可以看到一个按计划执行的、且阶段可控的软件开发过程。

- 已定义级：用于管理的和工程的软件过程均已文档化、标准化，并形成整个软件组织的标准软件过程。全部项目均采用与实际情况相吻合的、适当修改后的标准软件过程来进行操作。要求制订企业范围的工程化标准，而且无论管理还是工程开发都需要一套文档化的标准，并将这些标准集成到企业软件开发标准过程中去。所有开发的项目需根据这个标准过程，剪裁出项目适宜的过程，并执行这些过程。过程的剪裁不是随意的，在使用前需经过企业有关人员的批准。
- 已管理级：软件过程和产品质量有详细的度量标准。软件过程和产品质量得到了定量的认识和控制。已管理级的管理是量化的管理。所有过程需建立相应的度量方式，所有产品的质量（包括工作产品和提交给用户的产品）需有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品，量化控制将使软件开发真正变成为一个工业生产活动。
- 优化级：通过对来自过程、新概念和新技术等方面的各种有用信息的定量分析，能够不断地、持续地进行过程改进。如果一个企业达到了这一级，表明该企业能够根据实际的项目性质、技术等因素，不断调整软件生产过程以求达到最佳。

3．CMMI 综述

能力成熟度模型集成（Capability Maturity Model Integration，CMMI）是 CMM 模型的最新版本。早期的 CMMI（CMMI-SE/SW/IPPD）1.02 版本是应用于软件业项目的管理方法，SEI 在部分国家和地区开始推广和试用。随着应用的推广与模型本身的发展，演绎成为一种被广泛应用的综合性模型。2001 年 12 月，SEI 正式发布了 CMMI 1.1 版本。与原有的能力成熟度相比，CMMI 涉及面更广，专业领域覆盖软件工程、系统工程、集成产品开发和系统采购。据美国国防部资料显示，运用 CMMI 模型管理的项目，不仅降低了项目的成本，而且提高了项目的质量与按期完成率。

CMMI 可以看作把各种 CMM 集成到一个系列的模型中，CMMI 的基础源模型包括软件 CMM 2.0 版（草稿 C）、EIA-731 系统工程，以及集成化产品和过程开发 IPD CMM（IPD）0.98a 版。CMMI 也描述了 5 个不同的成熟度级别。

- 级别1（初始级）代表了以不可预测结果为特征的过程成熟度。过程包括了一些特别的方法、符号、工作和反映管理，成功主要取决于团队的技能。
- 级别2（已管理级）代表了以可重复项目执行为特征的过程成熟度。组织使用基本纪律进行需求管理、项目计划、项目监督和控制、供应商协议管理、产品和过程质量保证、配置管理，以及度量和分析。对于级别2而言，主要的过程焦点在于项目级的活动和实践。
- 级别3（严格定义级）代表了以组织内改进项目执行为特征的过程成熟度。强调级别3的关键过程域前后一致的、项目级的纪律，以建立组织级的活动和实践。
- 级别4（定量管理级）代表了以改进组织性能为特征的过程成熟度。4级项目的历史结果可用来交替使用，在业务表现的竞争尺度（成本、质量、时间）方面的结果是可预测的。
- 级别5（优化级）代表了以可快速进行重新配置的组织性能，与定量的、持续的过程改进为特征的过程成熟度。

4.6 软件项目管理

软件项目本身是复杂的，如果没有仔细地进行计划，复杂的项目是不可能成功的。一个计划良好的项目将受到有效的控制，进展明显，而参加该项目的人员都会得到支持以进行其工作。软件项目本身也具有风险性，如果没有有效的风险管理也是不能成功的。

通常软件工程项目的管理比其他工程项目的管理更困难，这是因为：

- 软件产品不可见。开发的进度及产品的质量是否符合要求不明显，比较难进行把握。
- 没有标准的软件过程。尽管近几年来“软件过程改进”领域有许多进步，但由于团队、人员的个性化因素，还不存在一个放之四海皆真理的标准化软件过程。
- 大型软件项目常常是一次性项目。由于这些项目都是“前无古人”的，因此缺乏可以借鉴的历史经验。

4.6.1 软件项目管理的内容

软件工程项目管理的对象是软件工程项目，它涉及整个软件开发过程。最核心的内容就是在成本、质量与进度之间做出平衡的取舍，如图 4-9 所示。

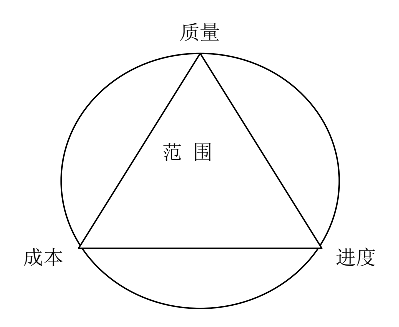


图 4-9 项目管理铁三角

抽象地说，其实软件项目管理也十分简单，主要包括 POIM 四个方面：即 Plan（计划）、Organize（组织）、Implement（实现）、Measurement（度量）。细化地说，主要包括如下一些主要的活动。

1．启动软件项目

通常，在启动软件项目之前，需要确定出项目的目标和范围，并根据这些内容来考虑解决方案，然后根据解决方案进行社会、经济、技术可行性分析，以确定是否启动项目。

然后再根据合理的、精确的成本估算，对其进行切实可行的任务分解，以及管理的进度安排，也就是完成最初的项目计划。

2．度量

在管理学中有一句名言：“如果不能够用数字来描述它，那么说明你还没有了解它。”在软件开发过程中也是一样，度量是帮助有效地定量管理的基础。度量的目的是为了把握软件工程过程的实际情况和它所生产的产品质量。

3．估算

制订项目计划是一个十分关键、重要的活动，但要想做出合理、有效的项目计划，就必须对需要的人力、项目的持续时间、成本做出相应的估算。

估算的基础是历史数据及经验模型。估算通常可以使用“自顶向下”和“自底向上”两种模型，在估算的过程中常见的经验模型包括 FP（功能点）和 COCOMO 系列。而要想获得更加准确、符合项目组队实际情况的估算值，就必须建立完整的历史项目资料库。

4．风险分析

正如 Tom Gilb 在其与软件工程管理相关的一本著作中说过:“如果谁不主动攻击风险，那么它们就会主动地攻击谁。”项目中风险无处不在，但却有很多项目管理者对此熟视无睹，从而带来了许多巨大的损失。在系统开发过程中，对潜在的风险进行分析，记录下来，按其发生的可能性和影响性进行排序，并制订相应的解决方案和预防措施。

试想，如果风险按你的预料“如期而至”时，你从容不迫地按照预算设定的解决方案来解决，将会给团队多大的信心？

5．进度安排

每一个软件项目都应该制订一个进度安排，而制订进度计划时需要考虑的问题包括：预先如何对进度进行计划？工作如何到位？如何识别定义的任务？如何监控任务的完成？如何设立分隔任务的里程碑？

进度安排通常建立在 WBS（工作任务分解）的基础上，对时间、人员、设备等资源进行统一的分配，根据估算的工作量进行计划。最常见的进度计划的工具包括甘特图、PERT 图等，而 Microsoft Project 则可以帮助使用者更好地应用这些工具进行进度的管理。

6．追踪和控制

正如拿破仑所说：“没有一场战争是按计划完成的，但没有一场战争没有计划。”这句话辩证地说明了计划与执行之间的关系。只有计划没有执行项目则不可能取得成功，因此需要对计划进行有效的追踪和控制，根据实际的执行情况对其进行有效的调整。

4.6.2　软件项目估算

在计算技术发展的早期，软件的成本在整个计算机系统的总成本中占的只是一个较小的百分比，因此即使软件项目估算有很大的误差，对整个项目而言影响还是相对较小的。但随着信息化的深入普及与发展，软件在整个项目中的比重也越来越大，因此如果无法精确地进行软件项目的估算，将会给整个计算机系统的预算带来很大的麻烦。

软件项目的估算策略包括“自顶向下”和“自底向上”两种，而估算的内容主要包括：软件规模估算、软件工作量估算及成本估算三个方面。

1．两种软件估算策略

不同的估算策略能够应用于不同的场合，也将带来完全不同的效果。

1）自顶向下估算法

自顶向下的估算法，通常是由项目经理自身或者是以项目经理为主的一个核心小组（通常包括分析师、构架师、主程序员等主要的角色）来完成的。其工作的特点是：先根据用户、决策者的要求，确定一个时间期限。然后根据这个时间期限进行分解，将开发工作进行对号入座，以获得一个可以满足这个期限的估算。

这种方式是一种通常采用的方法，但其并不能够有效地解决项目估算的问题，经常容易使得估算值与实际值产生很大的差异。

2）自底向上估算法

自底向上估算法则采用了一种完全不同的策略。首先在核心小组内进行头脑风暴，完成工作任务分解，直到每一个任务块都小到能够得出合理的估算为止（通常是两周以内的工作量）。然后将每个任务根据项目成员的技能特点、兴趣特长进行分配，并要求其对此做出估算。最后将这些估算值合并在一起，得到总的项目估算。

这种方式通常能够得到较为客观的、可操作的估算结果，而且还能够使得项目组成员主动地参与，并且通常能够对自己所做的承诺全力守信，从而为项目树立了一个良好的榜样。但由于其通常得到的值要远比预期的值大，时间更久，因此许多项目不能够有效地使用它。

2．软件规模估算

软件规模也就是需要完成的工作范围，这个估算结果是整个软件项目估算的基础。对于软件规模估算来说，最常见的是 LOC 和 FP 估算法。

1）LOC 估算法

LOC 是指估算软件的代码行数（Line Of Code），通常使用 KLOC（千代码行）为单位。这种估算的基础是将软件项目切分为一个个小模块，然后通过历史的项目经验数据，以及开发人员的经验，对其需要的 LOC 数进行估算。

2）FP 估算法

FP（功能点）是一种衡量工作量大小的单位，它的计算方法是：功能点=信息处理规模×技术复杂度。其中，技术复杂度=0.65+调节因子。

①信息处理规模

FP 方法通过外部输入数、外部输出数、外部查询数、内部逻辑文件数、外部接口文件数 5 个方面来衡量整个软件系统的信息处理规模。计算的方法如表 4-18 所示。

表 4-18　FP 基本功能点计算公式

	低	平均	高	说　明
Input	×3	×4	×6	Screen、From、etc.
Output	×4	×5	×7	Screen、Report、etc.
Inquire	×3	×4	×6	查询、处理请求
File	×7	×10	×15	内部所需的文件
Interface	×5	×7	×10	引用外部的数据

②技术复杂度

技术复杂度是从数据通信复杂度、分布式处理复杂度、性能复杂度、配置项负载复杂度、事务率复杂度、在线数据项复杂度、用户使用效率复杂度、在线更新复杂度、复杂处理复杂度、重用性复杂度、安装容易程度复杂度、操作容易程度复杂度、多个地点复杂度、修改容易程度复杂度 14 个方面进行微调。每个方面都根据其复杂程度，在 0～0.05 之间取值。将这些值全部相加就可以得到调节因子，再加上 0.65 就可以得到技术复杂度。

3．软件工作量估算

当通过 LOC 或 FP 估算获得了软件的规模之后，即可进行工作量的估算。工作量的单位通常是人月（对于大项目可以用人年，小项目则可以用人天）。从中不难得知，软件工作

量的获得应该是：规模/产能=工作量。

例如，如果估算出项目需要 5KLOC，而每个人一个月的产能是 1KLOC，那么很显然需要 5 个人月的时间来完成该项目。从中，也可以看出建立项目过程数据库，保存这些数字对于估算而言十分重要。

如果没有足够的可类比的项目数据做参照，那么也可以借助一些经验模型来进行估算，从而获得软件的工作量和进度的值。常见的经验模型包括大名鼎鼎的 COCOMO，以及相对较简单的 IBM 模型、普特南（Putnam）模型三种。下面主要介绍 COCOMO 模型。

COCOMO 模型是 TRW 公司开发的，是软件工作量估算的最有代表性的方法。根据考虑的因素的多少，详细程度的不同，可以将 COCOMO 模型分为三种：基本 COCOMO 模型、中间 COCOMO 模型和详细 COCOMO 模型。

- 基本COCOMO模型：基本COCOMO模型适用于快速、早期、粗数量级的软件成本估算，但由于未考虑硬件约束，人员素质和经验，现代化工具和技术的使用，以及对软件成本有着重大影响的其他已知项目属性之间的差异的作用（它们必然影响软件成本），其精确性必然有限。
- 中间COCOMO模型：中间COCOMO模型是与基本COCOMO模型兼容的，同时也是对基本COCOMO模型的扩展。它所具有的较高的精确性和详细程度使其更适于作为软件产品定义的更详细阶段中的成本估算。
- 详细COCOMO模型：在中间COCOMO模型的基础上，针对每一个影响因素按模块层、子系统层、系统层，做出三张工作量因素分级表，供不同层次的估算使用。

4．成本估算

当估算出工作量（人月数），以及知道的人员需求量、项目的持续时间之后，就可以进一步进行成本估算。也就是将人员的工资及相关管理费用之和，去乘以其所需要的时间，就可以得到人员成本。

最后再加上相关的资源成本（如软/硬件资源）、日常相关的其他开支成本等，就可以得到一个相对准确的成本估算值。

4.6.3 软件项目组织与计划

从项目经理的角度来看，整个项目实施的过程，就像驾车开往某地一样。首先要根据目的地的距离（类比项目的需求范围）、车况（类比团队的人员结构、平均产能）、时间的要求（完成期限）来制订合理的行驶计划（类比软件的项目计划）。

接着，就按照既定的计划执行。

在这个过程中，每隔一段时间检查一下进度情况，即通过观察里程碑来确定自己已经行驶过的路程（已完成的部分），并与计划进行比较，然后根据实际的情况，调整计划。例如，如果全长 300 千米，预计 3 小时完成，而 1 小时后，你发现由于路上遇到了一些问题，只行驶了 80 千米，那么，若要按计划完成，则必须提高时速。这就是监控和计划修正的过程。

另外，如果在行驶的过程中，你得到了新的指示，目的地发生了改变（类比项目的需求变更），那么也需要停下来，重新制订计划。在项目计划的制订过程中，不同的管理领域会用到多种不同的工具，在此主要介绍时间管理中的 PERT 分析与甘特图。

1．PERT 分析

计划评审技术（Program/Project Evaluation and Review Technique，PERT），是利用网络分析制订计划以及对计划予以评价的技术。它能协调整个计划的各道工序，合理安排人力、物力、时间、资金，加速计划的完成。在现代计划的编制和分析手段中，PERT 被广泛地使用，是现代项目管理的重要手段和方法。使用 PERT 解决问题，必须先掌握如下几个概念。

1）最早开始时间和最早完成时间

- 最早开始时间：一项活动的最早开始时间ES（Early Start Time）取决于它的所有紧前活动的完成时间。通过计算到该活动路径上所有活动的完成时间的和，可得到指定活动的ES。如果有多条路径指向此活动，则计算需要时间最长的那条路径。其计算分式如下。

$$ES=max\{\text{紧前活动的 }EF\}$$

- 最早完成时间：一项活动的最早完成时间EF（Early Finish Time）取决于该工作的最早开始时间和它的持续时间D，其计算公式如下。

$$EF= ES+D$$

2）最迟完成时间和最迟开始时间

- 最迟完成时间：在不影响项目完成时间的条件下，一项活动可能完成的最迟时间，简称为LF（Late Finish Time）。其计算公式如下。

$$LF=min\{\text{紧后活动的 }LS\}$$

- 最迟开始时间：在不影响项目完成时间的条件下，一项活动可能开始的最晚时间，简称为LS（Late Start Time）。其计算公式如下。

$$LS= LF-D$$

3）时差

- 总时差：即在不延误总工期的前提下，该活动的机动时间。一项活动的最早开始时间和最迟开始时间之间的差值就是该工作的总时差，简称为TF(Total Float Time)，计算公式如下。

$$TF= LS-ES$$

- 自由时差：在不影响紧后活动的最早开始时间前提下，某项活动的机动时间就是该项活动的自由时差，简称为FF（Free Float Time），它由该项活动的最早完成时间EF和它的紧后活动的最早开始时间决定。其计算公式如下。

$$FF=min\{\text{紧后活动的 }ES\}-EF$$

4）关键路径的确定

项目的关键路径是指能够决定项目最早完成时间的一系列活动。它是网络图中的最长路径，具有最少的浮动时间或时间差。尽管关键路径是最长的路径，但它代表了完成项目所需的最短时间。

如果关键路径上有一项或多项活动花费的时间超过了计划时间，那么整个项目进度就会拖延，除非项目经理采取了改进措施。下面以一个例题介绍在箭线图中关键路径的确定

及活动的最早开始时间、最早完成时间、最迟开始时间、最迟完成时间以及时差的计算。

【例题】某项目的箭线图如图 4-10 所示，计算活动 B、G、H 的最早开始时间、最早完成时间、最迟开始时间、最迟完成时间、总时差、自由时差，并确定关键路径和关键活动。假设活动 A 的最早开始时间为 0，活动 M 的最迟完成时间为 47。

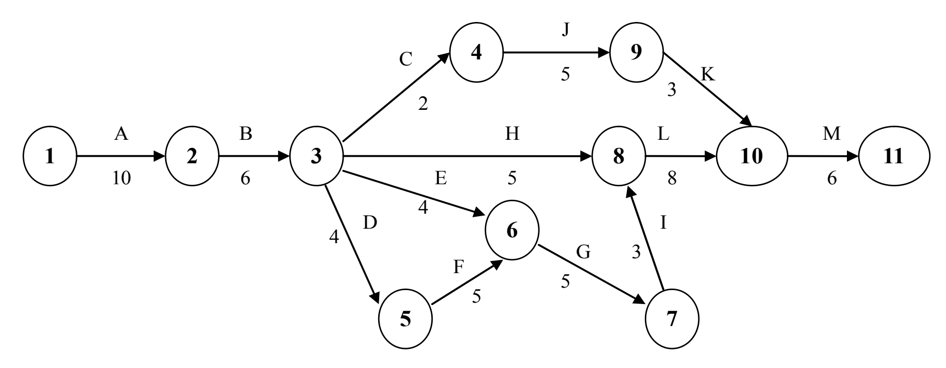


图 4-10　某项目箭线图

第一步，计算最早开始时间 ES 和最早完成时间 EF。

很容易看出，

对于活动 B：　ESB=10　　　　　　EFB=ESB+6=16

对于活动 H：　ESH=16　　　　　　EFH=ESH+5=21

对于活动 G：

其紧前活动有 E 和 F，而从图中可以看出，ESE=16，EFE=ESE+4=20，ESF=20，EFF=ESF+5=25，所以：ESG=*max*(EFE，EFF)=25，　　　　EFG=ESG+5=30。

第二步，计算最迟开始时间 LS 和最迟完成时间 LF。

在第二步计算时，应先计算最后一项活动的最迟完成时间，再计算最后一项活动的最迟开始时间，然后分别求出其紧前活动的最迟完成时间 LF 和最迟开始时间 LS，依此类推，直到求出全部活动的相关值。

在本题中，

活动 M：LFM=47，LSM=LFM−D=41

活动 L：LFL= LSM=41，LSL=LFL−D=33

因为 LFI= LSL=33，LSI=LFI−D=30，则活动 G：LFG= LSI=30，LSG=LFG−D=25

活动 H：LFH= LSL=33，LSH=LFH−D=28

按前面的计算方式可计算出：LSC=31，LSE=21，LSD=16，则

活动 B：LFB=*min*{ LSC ，LSH，LSE，LSD }=16，LSB=LFB−D=10

第三步，计算各项活动的总时差 TF 和自由时差 FF。

先按照第一步分别求出：ESC=16，ESE=16，ESD=16，ESI=30，ESL=33；

活动 B：TFB=LSB−ESB=0，FFB=*min*{ESC，ESH，ESE，ESD}−EFB=0

活动 G：TFG= LSG−ESG=0，FFG=ESI−EFG=0

活动 H：TFH= LSH−ESH=28-16=12，FFH=ESL−EFH=33−21=12

第四步，确定网络图关键路径和关键活动。

从网络图中可以找出 4 条从开始到完成的路径：A-B-C-J-K-M、A-B-H-L-M、A-B-E-G-I-L-M、A-B-D-F-G-I-L-M，这 4 条路径的长度分别为：32、35、42 和 47。则该项目的关键路径为：A-B-D-F-G-I-L-M。因此，关键活动为：A、B、D、F、G、I、L 和 M。

如果将该项目的所有的活动的相关数据全部求出，可以得出如下结论：关键活动的总时差和自由时差均为零，但是总时差和自由时差为零的活动不一定是关键活动。

对于前导图中的关键路径的确定及活动的最早开始时间、最早完成时间、最迟开始时间、最迟完成时间、时差的计算与箭线图类似。

2．甘特图

甘特图（Gantt Chart，GC），又称为横道图、条形图，它把计划和进度安排两种职能结合在一起，纵向列出项目活动，横向列出时间跨度。每项活动计划或实际的完成情况用横道线表示，横道图通过日历形式列出项目活动工期及其相应的开始和完成日期。图 4-11 所示为一个甘特图的实例。

甘特图的优点是简单、明了、直观，能较清楚地反映活动的开始和完成时间。但对于错综复杂、相互制约的各项活动间的逻辑依赖关系无法表示出来。

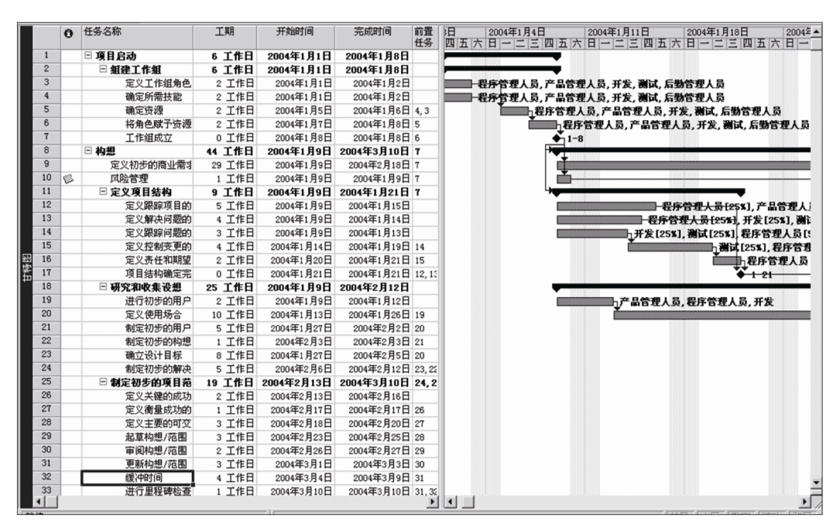


图 4-11　甘特图

4.6.4 风险管理

为什么要在软件项目中进行风险管理呢？一是关心未来，风险是否会导致软件项目失败；二是关心变化，在用户需求、开发、技术、目标机器，以及所有其他与项目有关的实体中会发生什么变化；三是必须解决选择问题，应当采用什么方法和工具，应当配备多少人力，在质量上强调到什么程度才满足要求。

风险管理通常包括 3 个主要活动：风险识别、风险估计和风险驾驭。

1．风险识别

可以用不同的方法对风险进行分类。从宏观上来看，风险可以分为项目风险、技术风险和商业风险。项目风险包括潜在的预算、进度、个人、资源、用户和需求方面的问题。技术风险包括潜在的设计、实现、接口、检验和维护方面的问题。而商业风险则主要来源于市场。

风险识别的重要工作就是将潜在的风险找到，并文档化。

2．风险估计

风险估计使用两种方法来估计每一种风险。一种方法是估计其发生的可能性；另一种方法是估计它可能带来的破坏性。然后根据这样的结果对其进行排列优先级，对于那种可能性大、破坏力也大的风险就应该更加重视，拟定相应的解决方案才能够有效地防范。

3．风险驾驭

风险驾驭是指利用某种技术，如原型化、软件自动化、软件心理学、可靠性工程学，以及某些项目管理方法等设法避开或转移风险。

第 5 章

数据库系统

数据库系统，是由数据库及其管理软件组成的系统。它是为适应数据处理的需要而发展起来的一种较为理想的数据处理系统，也是一个实际可运行的存储、维护和应用系统提供数据的软件系统，是存储介质、处理对象和管理系统的集合体。数据库技术是目前开发过程中不可回避的一个技术主题。本章将介绍数据库技术的发展史，并详细论述数据库规范化技术、SQL、数据库控制、数据仓库及分布式数据库的相关知识。

5.1 数据库管理系统的功能和特征

数据管理技术的发展大致经历了人工管理阶段（20 世纪 50 年代中期前）、文件系统阶段（20 世纪 50 年代后期到 60 年代中期）、数据库阶段（20 世纪 60 年代末到 70 年代末）和高级数据库技术阶段（20 世纪 80 年代初开始）。

数据库是长期存储在计算机内的、有组织的、可共享的数据的集合。

数据库管理系统（DBMS）是一种负责数据库的定义、建立、操作、管理和维护的软件系统。其目的是保证数据安全可靠，提高数据库应用的简明性和方便性。DBMS 的工作机理是把用户对数据的操作转化为对系统存储文件的操作，有效地实现数据库三级之间的转化。数据库管理系统的主要职能有：数据库的定义及建立、数据库的操作、数据库的控制、数据库的维护、故障恢复和数据通信。

数据库系统（DBS）是实现有组织地、动态地存储大量关联数据方便多用户访问的计算机软件、硬件和数据资源组成的系统。一个典型的数据库系统包括数据库、硬件、软件（应用程序）和数据库管理员（DBA）4 个部分。根据计算机的系统结构，DBS 可分成集中式、客户/服务器式、并行式和分布式 4 种。

与文件系统阶段相比，数据库技术的数据管理方式具有如下特点。

- 采用复杂的数据模型表示数据结构，数据冗余小，易扩充，实现了数据共享。
- 具有较高的数据和程序独立性，数据库的独立性有物理独立性和逻辑独立性。
- 数据库系统为用户提供了方便的用户接口。
- 数据库系统提供4个方面的数据控制功能，分别是并发控制、恢复、完整性和安全性。数据库中各个应用程序所使用的数据由数据库系统统一规定，按照一定的数据模型组织和建立，由系统统一管理和集中控制。
- 增加了系统的灵活性。

高级数据库技术阶段的主要标志是分布式数据库系统和面向对象数据库系统的出现。

集中式系统的弱点是随着数据量的增加，系统相当庞大，操作复杂，开销大，而且因为数据集中存储，大量的通信都要通过主机，造成拥挤。分布式数据库系统的主要特点是数据在物理上分散存储，在逻辑上是统一的。分布式数据库系统的多数处理就地完成，各地的计算机由数据通信网络相联系。

面向对象数据库系统是面向对象的程序设计技术与数据库技术相结合的产物。面向对象数据库系统的主要特点是具有面向对象技术的封装性和继承性，提高了软件的可重用性。

从目前的数据库系统来看，主要存在如下缺点：

- 采用静态数据模型，数据类型和操作简单、固定，只能处理短寿命事务。
- 不能适应计算机辅助设计、计算机辅助软件工程、图像处理、超文本、多媒体等新的应用。

数据库的未来发展趋势如下：

- 分布式数据管理。
- 支持面向对象的数据模型。
- 体系结构适应功能扩展，能处理复杂数据类型和长寿命事务，能和以前的数据库共存。
- 数据库技术与其他学科相结合（分布式数据库、并行数据库、多媒体数据库、Internet数据库、知识库、演绎数据库、主动数据库）。

5.2 数据库模型

数据库系统的设计目标是允许用户逻辑地处理数据，而不必涉及这些数据在计算机中是怎样存放的，在数据组织和用户应用之间提供某种程度的独立性。

5.2.1 数据库系统的三级结构

数据库技术中采用分级的方法，将数据库的结构划分为多个层次。最著名的是美国ANSI/SPARC 数据库系统研究组于 1975 年提出的三级划分法，如图 5-1 所示。

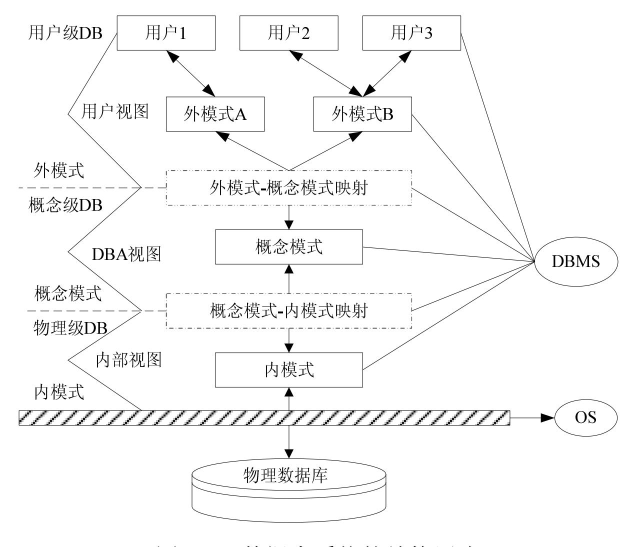


图 5-1 数据库系统的结构层次

数据库系统划分为 3 个抽象级：用户级、概念级和物理级。

1．用户级数据库

用户级数据库对应于外模式，是最接近于用户的一级数据库，是用户看到和使用的数据库，又称为用户视图。用户级数据库主要由外部记录组成，不同用户视图可以互相重叠，用户的所有操作都是针对用户视图进行的。

2．概念级数据库

概念级数据库对应于概念模式，介于用户级和物理级之间，是所有用户视图的最小并集，是数据库管理员看到和使用的数据库，又称为 DBA 视图。概念级数据库由概念记录组成，一个数据库可有多个不同的用户视图，每个用户视图由数据库某一部分的抽象表示所组成。一个数据库应用系统只存在一个 DBA 视图，它把数据库作为一个整体的抽象表示。概念级模式把用户视图有机地结合成一个整体，综合平衡考虑所有用户要求，实现数据的一致性，最大限度降低数据冗余，准确地反映数据间的联系。

3．物理级数据库

物理级数据库对应于内模式，是数据库的低层表示，它描述数据的实际存储组织，是最接近于物理存储的级，又称为内部视图。物理级数据库由内部记录组成。

5.2.2　数据库系统的三级模式

数据库系统的三级模式如图 5-1 所示。

1．概念模式

概念模式（模式、逻辑模式）用以描述整个数据库中数据库的逻辑结构，描述现实世界中的实体及其性质与联系，定义记录、数据项、数据的完整性约束条件及记录之间的联系，是数据项值的框架。

数据库系统概念模式通常还包含访问控制、保密定义、完整性检查等方面的内容，以及概念/物理之间的映射。

概念模式是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。一个数据库只有一个概念模式。

2．外模式

外模式（子模式、用户模式）用以描述用户看到或使用的那部分数据的逻辑结构，用户根据外模式用数据操作语句或应用程序去操作数据库中的数据。外模式主要描述组成用户视图的各个记录的组成、相互关系、数据项的特征、数据的安全性和完整性约束条件。

外模式是数据库用户（包括程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。一个数据库可以有多个外模式。一个应用程序只能使用一个外模式。

3．内模式

内模式是整个数据库的最低层表示，不同于物理层，它假设外存是一个无限的线性地址空间。内模式定义的是存储记录的类型、存储域的表示、存储记录的物理顺序，指引元、索引和存储路径等数据的存储组织。

内模式是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。一个数据库只有一个内模式。

4．三级模式的关系

- 模式是数据库的中心与关键。
- 内模式依赖于模式，独立于外模式和存储设备。
- 外模式面向具体的应用，独立于内模式和存储设备。
- 应用程序依赖于外模式，独立于模式和内模式。

5.2.3 数据库系统两级独立性

数据库系统两级独立性是指物理独立性和逻辑独立性。三个抽象级间通过两级映射（外模式/模式映射，模式/内模式映射）进行相互转换，使得数据库的三级形成一个统一的整体。

1．物理独立性

物理独立性是指用户的应用程序与存储在磁盘上的数据库中的数据是相互独立的。当数据的物理存储改变时，应用程序不需要改变。

物理独立性存在于概念模式和内模式之间的映射转换，说明物理组织发生变化时应用程序的独立程度。

2．逻辑独立性

逻辑独立性是指用户的应用程序与数据库中的逻辑结构是相互独立的。当数据的逻辑结构改变时，应用程序不需要改变。

逻辑独立性存在于外模式和概念模式之间的映射转换，说明概念模式发生变化时应用程序的独立程度。

值得注意的是，逻辑独立性比物理独立性更难实现。

5.3 数据模型

本节将介绍数据模型。

5.3.1 数据模型的分类

数据模型主要有两大类，分别是概念数据模型（实体联系模型）和基本数据模型（结构数据模型）。

概念数据模型是按照用户的观点来对数据和信息建模，主要用于数据库设计。概念模型主要用实体联系方法（Entity-Relationship Approach）表示，所以也称为 ER 模型。

基本数据模型是按照计算机系统的观点对数据和信息建模，主要用于 DBMS 的实现。基本数据模型是数据库系统的核心和基础。基本数据模型通常由数据结构、数据操作和完整性约束三部分组成。其中数据结构是对系统静态特性的描述，数据操作是对系统动态特性的描述，完整性约束是一组完整性规则的集合。

常用的基本数据模型有层次模型、网状模型、关系模型和面向对象模型。

层次模型用树型结构表示实体类型及实体间联系。层次模型的优点是记录之间的联系通过指针来实现，查询效率较高。层次模型的缺点是只能表示 $1:n$ 联系，虽然有多种辅助手段实现 $m:n$ 联系，但较复杂，用户不易掌握。由于层次顺序的严格和复杂，使得数据的

查询和更新操作很复杂，应用程序的编写也比较复杂。

网状模型用有向图表示实体类型及实体间联系。网状模型的优点是记录之间的联系通过指针实现，$m:n$ 联系也容易实现，查询效率高。其缺点是编写应用程序比较复杂，程序员必须熟悉数据库的逻辑结构。

关系模型用表格结构表达实体集，用外键表示实体间联系，其优点如下：

- 建立在严格的数学概念基础上。
- 概念单一（关系），结构简单、清晰，用户易懂易用。
- 存取路径对用户透明，从而数据独立性、安全性好，简化数据库开发工作。

关系模型的缺点主要是由于存取路径透明，查询效率往往不如非关系数据模型。

5.3.2 关系模型

先学习几个基本概念。

- 域：一组具有相同数据类型的值的集合。
- 笛卡儿积：给定一组域$D1$，$D2$，…，Dn，其中可以有相同的域。$D1$，$D2$，…，Dn的笛卡儿积为：

$D1 \times D2 \times \cdots \times Dn=\{(d1，d2，\cdots，dn)|dj \in Dj，j=1，2，\cdots，n\}$

其中每一个元素（$d1$，$d2$，…，dn）叫作一个 n 元组（简称为元组）。元组中的每一个值 dj 叫作一个分量。

- 关系：$D1 \times D2 \times \cdots \times Dn$的子集叫作在域$D1$，$D2$，…，$Dn$上的关系，表示为：

$$R(D1，D2，\cdots，Dn)$$

这里 R 表示关系的名字，n 是关系的目或度。

关系中的每个元素是关系中的元组，通常用 t 表示。关系是笛卡儿积的子集，所以关系也是一个二维表，表的每行对应一个元组，表的每列对应一个域。由于域可以相同，为了加以区分，必须为每列起一个名字，称为属性。

若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为候选码（候选键）。若一个关系有多个候选码，则选定其中一个为主码（主键）。主码的诸属性称为主属性。不包含在任何候选码中的属性称为非码属性（非主属性）。在最简单的情况下，候选码只包含一个属性。在最极端的情况下，关系模式的所有属性组是这个关系模式的候选码，称为全码。

关系可以有 3 种类型：基本关系（通常又称为基本表或基表）、查询表和视图表。基本表是实际存在的表，它是实际存储数据的逻辑表示。查询表是查询结果对应的表。视图表是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据。

基本关系具有如下 6 个特点。

- 列是同质的，即每一列中的分量是同一类型的数据，来自同一个域。
- 不同的列可出自同一个域，称其中的每一列为一个属性，不同的属性要给予不同的属性名。
- 列的顺序无所谓，即列的次序可以任意交换。
- 任意两个元组不能完全相同。但在大多数实际关系数据库产品中，例如Oracle等，

如果用户没有定义有关的约束条件，它们都允许关系表中存在两个完全相同的元组。

- 行的顺序无所谓，即行的次序可以任意交换。
- 分量必须取原子值，即每一个分量都必须是不可分的数据项。

关系的描述称为关系模式。一个关系模式应当是一个五元组。它可以形式化地表示为：$R(U，D，DOM，F)$。其中 R 为关系名，U 为组成该关系的属性名集合，D 为属性组 U 中属性所来自的域，DOM 为属性向域的映像集合，F 为属性间数据的依赖关系集合。关系模式通常可以简记为：$R(A1，A2，\cdots，An)$。其中 R 为关系名，$A1，A2，\cdots，An$ 为属性名。

关系实际上就是关系模式在某一时刻的状态或内容。也就是说，关系模式是型，关系是它的值。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。但在实际当中，常常把关系模式和关系统称为关系，读者可以从上下文中加以区别。

在关系模型中，实体及实体间的联系都是用关系来表示的。在一个给定的现实世界领域中，相应于所有实体及实体之间的联系的关系集合构成一个关系数据库。

关系数据库也有型和值之分。关系数据库的型也称为关系数据库模式，是对关系数据库的描述，是关系模式的集合。关系数据库的值也称为关系数据库，是关系的集合。关系数据库模式与关系数据库通常统称为关系数据库。

5.3.3 关系规范化理论

1．关系模式的存储异常问题

设有一个关系模式 R（SNAME，CNAME，TNAME TADDRESS），其属性分别表示学生姓名、选修的课程名、任课教师姓名和任课教师地址。仔细分析，这个模式存在如下存储异常的问题。

- 数据冗余：如果某门课程有100个学生选修，那么在R的关系中就要出现100个元组，这门课程的任课教师姓名和地址也随之重复出现100次。
- 修改异常：由于上述冗余问题，当需要修改这个教师的地址时，就要修改100个元组中的地址值，否则就会出现地址值不一致的现象。
- 插入异常：如果不知道听课学生名单，则这个教师的任课情况和家庭地址就无法进入数据库；否则就要在学生姓名处插入空值。
- 删除异常：如果某门课程的任课教师要更改，那么原来任课教师的地址将随之丢失。

因此，模式 R 虽然只有 4 个属性，但却是性能很差的模式。如果把 R 分解成如下两个关系模式：$R1$（SNAME，CNAME）和 $R2$（CNAME，TNAME，TADDRESS），则能消除上述提出的存储异常现象。

为什么会产生这些异常呢？与关系模式属性值之间的联系直接有关。在模式 R 中，学生与课程有直接联系，教师与课程有直接联系，而教师与学生无直接联系，这就产生了模式 R 的存储异常。因此，模式设计强调“每个联系单独表达”是一条重要的设计原则，把 R 分解成 $R1$ 和 $R2$ 是符合这条原则的。

2．函数依赖

设 $R(U)$是属性 U 上的一个关系模式，X 和 Y 是 U 的子集，r 为 R 的任一关系，如果对于 r 中的任意两个元组 u,v，只要有 $u[X]=v[X]$，就有 $u[Y]=v[Y]$，则称 X 函数决定 Y，或称 Y

函数依赖于 X，记为 $X \to Y$。

从函数依赖的定义可以看出，如果有 $X \to U$ 在关系模式 R（U）上成立，并且不存在 X 的任一真子集 X'，使 $X' \to U$ 成立，那么称 X 是 R 的一个候选键。也就是 X 值唯一决定关系中的元组。由此可见，函数依赖是键概念的推广，键是一种特殊的函数依赖。

在 $R(U)$中，如果 $X \to Y$，并且对于 X 的任何一个真子集 X'，都有 $X' \to Y$ 不成立，则称 Y 对 X 完全函数依赖。若 $X \to Y$，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖。

在 $R(U)$中，如果 $X \to Y$（Y 不是 X 的真子集），且 $Y \to X$ 不成立，$Y \to Z$，则称 Z 对 X 传递函数依赖。

设 U 是关系模式 R 的属性集，F 是 R 上成立的只涉及 U 中属性的 FD 集，则有如下 3 条推理规则。

- 自反性：若 $Y \subseteq X \subseteq U$，则$X \to Y$在$R$上成立。
- 增广性：若$X \to Y$在R上成立，且 $Z \subseteq U$，则$XZ \to YZ$在R上成立。
- 传递性：若$X \to Y$和$Y \to Z$在R上成立，则$X \to Z$在R上成立。

这里 XZ，YZ 等写法表示 $X \cup Z$，$Y \cup Z$。上述 3 条推理规则是函数依赖的一个正确的和完备的推理系统。根据上述 3 条规则还可以推出其他 3 条常用的推理规则。

- 并规则：若$X \to Y$和$X \to Z$在R上成立，则$X \to YZ$在R上成立。
- 分解规则：若$X \to Y$在R上成立，且 $Z \subseteq Y$，则$X \to Z$在R上成立。
- 伪传递规则：若$X \to Y$和$WY \to Z$在R上成立，则$WX \to Z$在R上成立。

在关系模式 $R(U,F)$中被 F 逻辑蕴含的函数依赖全体叫作 F 的闭包，记作 F^+。

设 F 为属性集 U 上的一组函数依赖，X 是 U 的子集，那么相对于 F 属性集 X 的闭包用 X^+表示，它是一个从 F 集使用推理规则推出的所有满足 $X \to A$ 的属性 A 的集合：

X^+={属性 $A|X \to A$ 在 F^+中}

如果 $G^+=F^+$，就说函数依赖集 F 覆盖 G（F 是 G 的覆盖，或 G 是 F 的覆盖），或 F 与 G 等价。

如果函数依赖集 F 满足下列条件，则称 F 为一个极小函数依赖集，也称为最小依赖集或最小覆盖。

- F中任一函数依赖的右部仅含有一个属性。
- F中不存在这样的函数依赖$X \to A$，使得F与$F-\{X \to A\}$等价。
- F中不存在这样的函数依赖$X \to A$，X有真子集Z使得$F-\{X \to A\} \cup \{Z \to A\}$与$F$等价。

3．范式

- 第一范式（1NF）：如果关系模式R的每个关系r的属性值都是不可分的原子值，那么称R是第一范式的模式，r是规范化的关系。关系数据库研究的关系都是规范化的关系。
- 第二范式（2NF）：若关系模式R是1NF，且每个非主属性完全函数依赖于候选键，那么称R是2NF模式。
- 第三范式（3NF）：如果关系模式R是1NF，且每个非主属性都不传递依赖于R的候选码，则称R是3NF。

- BC范式（BCNF）：若关系模式R是1NF，且每个属性都不传递依赖于R的候选键，那么称R是BCNF模式。

上述 4 种范式之间有如下联系：1NF$\supset$2NF$\supset$3NF$\supset$BCNF。

4．关系模式分解

如果某关系模式存在存储异常问题，则可通过分解该关系模式来解决问题。把一个关系模式分解成几个子关系模式，需要考虑的是该分解是否保持函数依赖，是否是无损连接。

无损连接分解的形式定义如下：设R是一个关系模式，F是R上的一个函数依赖（FD）集。R分解成数据库模式δ={$R1$，…，Rk}。如果对R中每一个满足F的关系r都有下式成立：

$$r = \pi_{R1}(r) \bowtie \pi_{R2}(r) \bowtie \cdots \bowtie \pi_{Rk}(r)$$

那么称分解δ相对于F是“无损连接分解”，否则称为“损失连接分解”。

下面是一个很有用的无损连接分解判定定理。

设ρ=｛$R1$，$R2$｝是R的一个分解，F是R上的 FD 集，那么分解ρ相对于F是无损分解的充分必要条件是（$R1 \cap R2$）→（$R1-R2$）或（$R1 \cap R2$）→（$R2-R1$）。

设数据库模式δ={$R1$，…，Rk}是关系模式R的一个分解，F是R上的 FD 集，δ中每个模式Ri上的 FD 集是Fi。如果{$F1$，$F2$，…，Fk}与F是等价的（即相互逻辑蕴涵），那么称分解δ保持 FD。如果分解不能保持 FD，那么δ的实例上的值就可能有违反 FD 的现象。

5.4 数据操作

在关系数据库中，数据操作主要包括查询和更新两大类。关系数据语言有关系代数语言，关系演算语言，以及具有关系代数和关系演算双重特点的语言三种。其中关系演算语言又包括元组关系演算语言和域关系演算语言。

5.4.1 集合运算

传统的集合运算是二目运算，包括并、交、差、广义笛卡儿积 4 种运算。

1．并

设关系R和S具有相同的关系模式，R和S的并是由属于R或属于S的元组组成的集合，记为$R \bigcup S$。形式定义如下：

$$R \bigcup S \equiv \{t \mid t \in R \vee t \in S\}$$

式中t是元组变量（下同）。显然，$R \cup S$=$S \cup R$。

2．差

关系R和S具有相同的关系模式，R和S的差是由属于R但不属于S的元组组成的集合，记为R–S。形式定义如下：

$$R - S \equiv \{t \mid t \in R \wedge t \notin S\}$$

3．交

关系 R 和 S 具有相同的关系模式，R 和 S 的交是由既属于 R 又属于 S 的元组组成的集合，记为 $R \cap S$。形式定义如下：

$$R \cap S \equiv \{t \mid t \in R \wedge t \in S\}$$

显然，$R \cap S = R - (R - S)$，或者 $R \cap S = S - (S - R)$。

4．笛卡儿积

设关系 R 和 S 元数分别为 r 和 s。R 和 S 的笛卡儿积是一个 $r+s$ 元的元组集合，每个元组的前 r 个分量来自 R 的一个元组，后 s 个分量来自 S 的一个元组，记为 $R \times S$。形成定义如下：

$$R \times S \equiv \{t \mid t = < t_r, t_s > \wedge t_r \in R \wedge t_s \in S\}$$

若 R 有 m 个元组，S 有 n 个元组，则 $R \times S$ 有 $m \times n$ 个元组。

5．集合运算实例

例如，设关系 R 和 S 如表 5-1 所示。则 $R \cup S$ 与 $R \cap S$ 如表 5-2 所示，$R - S$ 和 $S - R$ 如表 5-3 所示，$R \times S$ 如表 5-4 所示。

表 5-1　关系 *R* 和 *S*

R 关系				*S* 关系		
A1	A2	A3		A1	A2	A3
a	b	c		a	b	a
b	a	d		b	a	d
c	d	d		c	d	d
d	f	g		d	s	c

表 5-2　*R*∪*S* 与 *R*∩*S*

R∪S				R∩S		
A1	A2	A3		A1	A2	A3
a	b	c		b	a	d
b	a	d		c	d	d
c	d	d				
d	f	g				
a	b	a				
d	s	c				

表 5-3　*R*－*S* 和 *S*－*R*

R－S				S－R		
A1	A2	A3		A1	A2	A3
a	b	c		a	b	a
d	f	g		d	s	c

表 5-4　$R \times S$

A1	A2	A3	A1	A2	A3
a	b	c	a	b	a
b	a	d	a	b	a
c	d	d	a	b	a
d	f	g	a	b	a
a	b	c	b	a	d
b	a	d	b	a	d
c	d	d	b	a	d
d	f	g	b	a	d
a	b	c	c	d	d
b	a	d	c	d	d
c	d	d	c	d	d
d	f	g	c	d	d
a	b	c	D	s	c
b	a	d	d	s	c
c	d	d	d	s	c
d	f	g	d	s	c

5.4.2　关系运算

在 5.4.1 节的集合运算基础上，关系数据库还有一些专门的运算，主要有投影、选择、连接、除法和外连接。它们是关系代数最基本的操作，也是一个完备的操作集。在关系代数中，由 5 种基本代数操作经过有限次复合的式子称为关系代数运算表达式。表达式的运算结果仍是一个关系。可以用关系代数表达式表示各种数据查询和更新处理操作。

1．投影

投影操作用于从关系 R 中选择出若干属性列组成新的关系，该操作对关系进行垂直分割，消去某些列，并重新安排列的顺序，再删去重复元组。记为：

$$\pi_A(R) \equiv \{t[A] \mid t \in R\}$$

其中 A 为 R 的属性列。

2．选择

选择操作在关系 R 中选择满足给定条件的所有元组，记为：

$$\sigma_F(R) \equiv \{t \mid t \in R \wedge F(t) = true\}$$

其中 F 表示选择条件，是一个逻辑表达式（逻辑运算符+算术表达式 ）。选择运算是从行的角度进行的运算。

3．θ 连接

θ 连接从两个关系的笛卡儿积中选取属性间满足一定条件的元组，记为：

$$R \underset{A\theta B}{\bowtie} S \equiv \{t_r t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A] \theta t_s[B]\}$$

其中 A 和 B 分别为 R 和 S 上度数相等且可比的属性组。θ 为“=”的连接，称为等值连接，记为：

$$R \underset{A=B}{\bowtie} S \equiv \{t_r t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B]\}$$

如果两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉，则称为自然连接，记为：

$$R \bowtie S \equiv \{t_r t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B]\}$$

4．除法

设两个关系 R 和 S 的元数分别为 r 和 s（设 $r > s > 0$），那么 $R \div S$ 是一个（$r-s$）元的元组的集合。$R \div S$ 是满足下列条件的最大关系：其中每个元组 t 与 S 中每个元组 u 组成新元组 $<t,u>$ 必在关系 R 中。其具体计算公式如下：

$$R \div S = \pi_{1,2,\cdots,r-s}(R) - \pi_{1,2,\cdots,r-s}((\pi_{1,2,\cdots,r-s}(R) \times S) - R)$$

5．外连接

两个关系 R 和 S 进行自然连接时，选择两个关系 R 和 S 公共属性上相等的元组，去掉重复的属性列构成新关系。这样，关系 R 中的某些元组有可能在关系 S 中不存在公共属性值上相等的元组，造成关系 R 中这些元组的值在运算时舍弃了；同样关系 S 中的某些元组也可能舍弃。为此，扩充了关系运算左外连接、右外连接和完全外连接。

- 左外连接：R和S进行自然连接时，只把R中舍弃的元组放到新关系中。
- 右外连接：R和S进行自然连接时，只把S中舍弃的元组放到新关系中。
- 完全外连接：R和S进行自然连接时，只把R和S中舍弃的元组都放到新关系中。

6．关系运算实例

设两个关系模式 R 和 S 如表 5-5 所示，则 $\pi_{1,2}(R)$ 的结果如表 5-6 所示，$\sigma_{1>2}(R)$ 的结果如表 5-7 所示，$R \bowtie S$ 的结果如表 5-8 所示，R 与 S 的左外连接如表 5-9 所示，R 与 S 的右外连接如表 5-10 所示，R 与 S 的完全外连接如表 5-11 所示。

表 5-5　关系 *R* 和 *S*

R 关系				*S* 关系		
A1	A2	A3		A1	A2	A4
a	b	c		a	z	a
b	a	d		b	a	h
c	d	d		c	d	d
d	f	g		d	s	c

表 5-6　对关系 *R* 求投影操作

A1	A2
a	b
b	a
c	d
d	f

表 5-7　对关系 *R* 求选择操作

A1	A2	A3
b	a	D

表 5-8　对关系 *R* 和 *S* 的自然连接

A1	A2	A3	A4
B	a	d	h
C	d	d	d

表 5-9　*R* 与 *S* 的左外连接

A1	A2	A3	A4
A	b	c	null
B	a	d	h
C	d	d	d
D	f	g	null

表 5-10　*R* 与 *S* 的右外连接

A1	A2	A3	A4
A	z	null	a
B	a	d	h
C	d	d	d
D	s	null	c

表 5-11　*R* 与 *S* 的完全外连接

A1	A2	A3	A4
A	b	c	null
B	a	d	h
C	d	d	d
D	f	g	null
A	z	null	a
D	s	null	c

5.5　数据库语言

下面简单地介绍标准化数据库查询语言 SQL。

SQL 语言由 Boyce 和 Chamberlin 于 1974 年提出，1975～1979 年，IBM San Jose Research Lab 的关系数据库管理系统原型 System R 实施了这种语言。SQL-86 是第一个 SQL 标准，后续的有 SQL-89、SQL-92（SQL2）、SQL-99（SQL3）等。现在大部分 DBMS 产品都支持 SQL，但每个产品在具体使用时又有方言，支持程度不同。

SQL 的特点主要体现在如下几个方面。

- 集数据定义语言、数据操纵语言、数据控制语言的功能于一体，语言风格统一。

- 存取路径的选择及SQL语句的操作过程由系统自动完成，减轻了用户负担，提高了数据独立性。
- 采用集合的操作方式。
- 既是自含式语言（联机交互），又是嵌入式语言（宿主语言）。
- 语言简捷，易学易用。只有10个动词（SELECT、CREATE、DROP、ALTER、INSERT、UPDATE、DELETE、GRANT、REVOKE、MODIFY）。

5.5.1 数据定义

1. 定义基本表

SQL 语言使用动词 CREATE 定义基本表，其具体语法格式如下：

```
CREATE TABLE <表名>
    (<列名><数据类型>[列级完整性约束条件][,
    <列名><数据类型>[列级完整性约束条件]][,
    <表级完整性约束条件>]);
```

例如，建立一个学生表 Student，它由学号 Sno、姓名 Sname、性别 Ssex、年龄 Sage、所在系 Sdept 5 个属性组成。其中学号不能为空，值是唯一的，并且姓名取值也唯一。

```
CREATE TABLE Student
    (Sno CHAR(5) NOT NULL UNIQUE,
    Sname CHAR(20) UNIQUE,
    Ssex CHAR(2),
    Sage INT,
    Sdept CHAR(15));
```

2. 修改基本表

修改基本表的命令格式如下：

```
ALTER TABLE <表名>
    [ADD <新列名><数据类型>[完整性约束]]
    [DROP <完整性约束名>]
    [MODIFY <列名><数据类型>];
```

例如，向 Student 表增加“入学时间”列，其数据类型为日期型。SQL 命令如下：

```
ALTER TABLE Student Add Scome Date;
```

3. 删除基本表

```
DROP TABLE <表名>
```

例如，要删除 Student 表的命令为：

```
DROP TABLE Student;
```

注意：基本表一旦删除，表中的数据、表上建立的索引和视图都将自动被删除。

4. 建立索引

建立索引的命令格式如下：

```
CREATE [Unique][Cluster]INDEX <索引名>
     ON <表名>(<列名>[<次序>][,<列名>[<次序>]]…);
```

其中<次序>可以为 ASC（升序，默认）、DESC（降序）。

Unique：每一个索引值只对应唯一的数据记录。

Cluster：聚簇索引，即索引项的顺序与表中记录的物理顺序一致。

例如，要在 Student 表的 Sname 列上建立一个聚簇索引，并按升序排列的命令为：

```
CREATE Cluster INDEX Stuname ON Student(Sname);
```

5．删除索引

删除索引的 SQL 命令格式如下：

```
DROP INDEX <索引名>
```

例如，要删除 Student 表的索引 Stuname 的命令为：

```
DROP INDEX Stuname;
```

5.5.2　数据查询

在 SQL 语言中，只提供了一个动词 SELECT 用来进行数据查询操作，但这个动词的参数十分复杂，且能嵌套使用，所以，考试时往往就考这个功能。其通用格式如下：

```
SELECT [All | Distinct] <目标列表达式>[,<目标列表达式>]…
FROM <表名或视图名>[, <表名或视图名>]…
[WHERE <条件表达式>]
[GROUP BY <列名 1>[HAVING <条件表达式>]]
[ORDER BY <列名 2>[ASC | DESC]];
```

1．单表查询

下面主要通过一些例子来说明 SELECT 语句的使用。假设有上述的 Student 表，还有课程表 Course（Cno，Cname，Credit，Cpno）和选修表 Sc（Sno，Cno，Grade）。其中 Cno 为课程号，Cname 为课程名称，Cpno 为先修课程号，Credit 为学分，Grade 为成绩。

- 查询全体学生的学号与姓名的命令格式为：

```
SELECT Sno,Sname
FROM student;
```

- 查询全体男学生的详细记录的命令格式为：

```
SELECT *
FROM student
WHERE Ssex='男';
```

- 查询所有年龄大于21岁的学生的姓名、出生年份和所有系，要求用小写字母表示所有系名。其格式如下：

```
SELECT Sname, 'Year of Birth:',2004 - Sage,lower(Sdept)
FROM student
WHERE Sage>21;
```

- 查询IS系、MA系和CS系学生的姓名和性别的命令格式为：

```
SELECT Sname,Sex
FROM student
WHERE Sdept In('IS','MA','CS');
```

- 查询名字中第2个字为“阳”的学生的姓名、学号的命令格式为：

```
SELECT Sname,Sno
FROM student
WHERE Sname LIKE '_ _阳%';
```

其中的“_”代表一个字符，而“%”代表 0 到若干个字符。

- 查询DB_Design课程的课程号和学分的命令格式为：

```
SELECT Cno,credit
FROM Course
WHERE Cname LIKE 'DB\_Design' Escape '\';
```

- 查询选修了3号课程的学生的学号及成绩，查询结果按分数的降序排列所有有成绩的学生学号和课程号。

```
SELECT Sno,Grade
FROM SC
WHERE Cno='3'
ORDER BY Grade DESC;
```

在 SQL 语言中，也可以使用集函数：

```
Count([Distinct|All]*)：统计元组个数；
Count([Distinct|All]<列名>)：统计一列中值的个数；
Sum([Distinct|All]<列名>)：计算一列值的总和；
Avg([Distinct|All]<列名>)：计算一列值的平均值；
max([Distinct|All]<列名>)：求一列值中的最大值；
Min([Distinct|All]<列名>)：求一列值中的最小值。
```

- 求各个课程号及相应的选课人数。

```
SELECT Cno,Count(Sno)
FROM SC
GROUP BY Cno;
```

2．连接查询

- 查询每个学生及其选修课程的情况。

```
SELECT Student.*,SC.*
FROM Student,SC
WHERE Student.Sno= SC.Sno;
```

- 查询每一门课程的间接选修课。

```
SELECT F.Cno,S.Cpno
FROM Course F,Course S
WHERE F.Cpno = S.Cno;
```

其中的 F 和 S 称为 course 的别名。

- 查询每个学生及其选修课程的情况。

```
SELECT Student.Sno,Sname,Ssex,Sage,Cno,Grade
FROM Student Left Outer Join SC
ON Student.Sno = SC.Sno;
```

- 查询每个学生的学号、姓名、选修的课程名称及成绩。

```
SELECT Student.Sno,Sname,Cname,Grade
FROM Student,SC,Course
WHERE Student.Sno=SC.Sno And SC.Cno=Course.Cno;
```

3．嵌套查询

- 查询与“刘晨”在同一系学习的学生。

```
SELECT Sno,Sname
FROM Student
WHERE Sdept IN
    (SELECT Sdept
    FROM Student
    WHERE Sname='刘晨');
```

- 查询选修了课程名为信息系统（MIS）的学生学号和姓名。

```
SELECT Sno,Sname
FROM Student
WHERE Sno IN
     (Select Sno
    FROM SC
    WHERE Cno IN
        (SELECT Cno
        FROM Course
        WHERE Cname='MIS'));
```

- 查询其他系中比信息系某一个学生年龄小的学生姓名和年龄。

```
SELECT Sname, Sage
FROM Student
     WHERE Sage < Any
    (SELECT Sage
    FROM Student
    WHERE Sdept='IS');
```

或者：

```
SELECT Sname, Sage
FROM Student
     WHERE Sage <
         (SELECT Max(Sage)
          FROM Student
          WHERE Sdept='IS')
         AND Sdept <> 'IS';
```

- 查询没有选修1号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE Not Exists
    (SELECT *
    FROM SC
    WHERE Sno=Student.Sno And Cno='1');
```

- 查询至少选修了95002选修表的全部课程的学生的学号。

```
SELECT Distinct Sno
FROM SC SCX
WHERE Not Exists
    (SELECT *
    FROM SC SCY
    WHERE SCY.Sno='95002'
       And Not Exists
```

```
        (SELECT *
          FROM SC SCZ
          WHERE SCZ.Sno=SCX.Sno And SCZ.Cno=SCY.Cno));
```

4．集合查询

例如，要查询计算机系的学生及年龄不大于 19 岁的学生的命令格式为：

```
SELECT *
FROM Student
WHERE Sdept='CS'
UNION
SELECT *
FROM Student
WHERE Sage<=19;
```

5.5.3　数据更新

1．插入数据

插入单个元组的命令格式为：

```
INSERT INTO <表名>[(<属性列 1>[,<属性列 2>…])
VALUES (<常量 1>[,<常量 2>]…)
```

例如，将一个新学生记录（95020，陈冬，男，IS，18）插入到 Student 表中。

```
INSERT INTO Student
VALUES ('95020', '陈冬', '男', 'IS',18);
```

2．修改数据

修改数据的命令格式为：

```
UPDATE <表名>
SET <列名 1>=<表达式 1>[,<列名 2>=<表达式 2>]…
[WHERE <条件>]
```

例如，将学号为 95001 的学生的年龄改为 22 岁。

```
UPDATE Student
SET Sage=22
WHERE Sno='95001';
```

3．删除数据

删除表中数据的命令格式为：

```
DELETE FROM <表名>
[WHERE <条件>]
```

例如，删除学号为 95019 的学生记录为：

```
DELETE FROM Student
WHERE Sno='95019';
```

5.5.4　视图

视图不真正存在数据，只是把定义存于数据字典，在对视图进行查询时，才按视图的定义从基本表中将数据查出。若一个视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了码，则这个视图称为行列子集视图。

在 DBMS 中，视图的作用如下：

- 简化用户的操作。
- 使用户能从多种角度看待同一数据。
- 对重构数据库提供了一定程度的逻辑独立性。
- 能够对机密数据提供安全保护。

1．定义视图

建立视图的命令格式如下：

```
CREATE VIEW <视图名>[(<列名>[,<列名>]…)]
AS
子查询
[With Check Option]
```

其中 With Check Option 表示对视图进行 Update、Insert 和 Delete 操作时，要保证更新、插入或删除的行满足视图定义中的谓词条件。

例如，建立信息系学生的视图：

```
CREATE VIEW IS_Student
AS
SELECT Sno,Sname,Sage
FROM Student
WHERE Sdept='IS'
With Check Option;
```

2．删除视图

删除视图的命令格式为：

```
DROP 视图名
```

例如，要删除视图 IS_S1：

```
DROP VIEW IS_S1;
```

3．查询视图

因为视图没有真实数据，所以，对视图的查询要转换为对相应表的查询，这个过程称为视图消解，视图消解过程由 DBMS 自动完成。

例如，在信息系学生的视图中找出年龄小于 20 岁的学生：

```
SELECT Sno,Sage
FROM IS_Student
WHERE Sage<20;
```

上述语句等价于：

```
SELECT Sno,Sage
FROM Student
WHERE Sage<20 And Sdept='IS';
```

4．更新视图

更新视图就是对相应表的更新。例如，将信息系学生视图 IS_Student 中学号为 95002 的学生姓名改为“刘辰”：

```
UPDATE  IS_Student
SET Sname='刘辰'
WHERE Sno='95002';
```

上述语句等价于：

```
UPDATE Student
SET Sname='刘辰'
WHERE Sno='95002' And Sdept='IS';
```

5.5.5 数据控制

1．授权

授权的命令格式如下：

```
GRANT <权限>[,<权限>]…
[ON <对象类型><对象名>]
TO <用户>[,<用户>]… [With Grant Option]
```

例如，把对 Student 表和 Course 表的全部操作权授予用户 U2 和 U3：

```
GRANT All Privileges
ON Table Student，Course
TO U2，U3;
```

又如，把对表 SC 的 Insert 权限授予 U5 用户，并允许 U5 将此权限再授予其他用户：

```
GRANT Insert
ON Table SC
TO U5 With Grant Option;
```

2．收回授权

收回授权的命令格式如下：

```
REVOKE <权限>[,<权限>]…
[ON <对象类型><对象名>]
FROM <用户>[,<用户>]…
```

例如，把用户 U4 修改学生学号的权限收回：

```
REVOKE Update(Sno),Select
ON Table Student
FROM U4;
```

5.6 数据库的控制功能

本节将介绍数据库的控制功能。

5.6.1 并发控制

数据库管理系统运行的基本工作单位是事务，事务是用户定义的一个数据库操作序列，这些操作序列要么全做，要么全不做，是一个不可分割的工作单位。事务具有如下特性。

- 原子性（Atomicity）：数据库的逻辑工作单位。
- 一致性（Consistency）：使数据库从一个一致性状态变到另一个一致性状态。
- 隔离性（Isolation）：不能被其他事务干扰。
- 持续性（永久性）（Durability）：一旦提交，改变就是永久性的。

事务通常以 BEGIN TRANSACTION（事务开始）语句开始，以 COMMIT 或 ROLLBACK 语句结束。COMMIT 称为“事务提交语句”，表示事务执行成功地结束。ROLLBACK 称为“事务回退语句”，表示事务执行不成功地结束。从终端用户来看，事务是一个原子，是不可分割的操作序列。事务中包括的所有操作要么都做，要么都不做（就效果而言）。事务不应该丢失或被分割完成。

在多用户共享系统中，许多事务可能同时对同一数据进行操作，称为“并发操作”，此时数据库管理系统的并发控制子系统负责协调并发事务的执行，保证数据库的完整性不受破坏，同时避免用户得到不正确的数据。

数据库的并发操作带来的问题：丢失更新问题，不一致分析问题（读过时的数据），依赖于未提交更新的问题（读了“脏”数据）。这三个问题需要 DBMS 的并发控制子系统来解决。

处理并发控制的主要方法是采用封锁技术。有两种封锁，X 封锁和 S 封锁。

- 排他型封锁（简称X封锁）：其含义是如果事务T对数据A（可以是数据项、记录、数据集以至整个数据库）实现了X封锁，那么只允许事务T读取和修改数据A，其他事务要等事务T解除X封锁以后，才能对数据A实现任何类型的封锁。可见X封锁只允许一个事务独锁某个数据，具有排他性。
- 共享型封锁（简称S封锁）：X封锁只允许一个事务独锁和使用数据，要求太严。需要适当从宽，例如，可以允许并发读，但不允许修改，这就产生了S封锁概念。S封锁的含义是如果事务T对数据A实现了S封锁，那么允许事务T读取数据A，但不能修改数据A，在所有S封锁解除之前决不允许任何事务对数据A实现X封锁。

在多个事务并发执行的系统中，主要采取封锁协议来进行处理。

- 一级封锁协议：事务T在修改数据R之前必须先对其加X封锁，直到事务结束才释放。一级封锁协议可防止丢失修改，并保证事务T是可恢复的。但不能保证可重复读和不读“脏”数据。
- 二级封锁协议：一级封锁协议加上事务T在读取数据R之前先对其加S锁，读完后即可释放S锁。二级封锁协议可防止丢失修改，还可防止读“脏”数据。但不能保证可重复读。
- 三级封锁协议：一级封锁协议加上事务T在读取数据R之前先对其加S锁，直到事务结束才释放。三级封锁协议可防止丢失修改、防止读“脏”数据与防止数据重复读。
- 两段锁协议：所有事务必须分两个阶段对数据项加锁和解锁。其中扩展阶段是在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；收缩阶段是在释放一个封锁之后，事务不能再申请和获得任何其他封锁。若并发执行的所有事务均遵守两段封锁协议，则对这些事务的任何并发调度策略都是可串行化的。遵守两段封锁协议的事务可能发生死锁。

下面讨论封锁的粒度。所谓封锁的粒度即是被封锁数据目标的大小，在关系数据库中封锁粒度有属性值、属性值集、元组、关系、某索引项（或整个索引）、整个关系数据库、物理页（块）等几种。

封锁粒度小则并发性高，但开销大。封锁粒度大则并发性低但开销小，综合平衡照顾不同需求以合理选取适当的封锁的粒度是很重要的。

采用封锁的方法固然可以有效防止数据的不一致性，但封锁本身也会产生一些麻烦，最主要的就是“死锁”（deadlock）问题。所谓死锁即是多个用户申请不同封锁，由于申请者均拥有一部分封锁权而又需等待另外用户拥有的部分封锁而引起的永无休止的等待。一般讲，死锁是可以避免的，目前采用的办法有如下几种。

- 预防法：此种方法是采用一定的操作方式以保证避免死锁的出现，顺序申请法、一次申请法等即是此类方法。所谓顺序申请法，即是对封锁对象按序编号，用户申请封锁时必须按编号顺序（从小到大或反之）申请，这样能避免死锁发生。所谓一次申请法，即是用户在一个完整操作过程中必须一次性申请它所需要的所有封锁，并在操作结束后一次性归还所有封锁，这样也能避免死锁的发生。
- 死锁的解除法：此种方法是允许产生死锁，并在死锁产生后通过解锁程序以解除死锁。使用这种方法需要有两个程序，一个是死锁检测程序，用它来测定死锁是否发生；另一个是解锁程序，一旦经测定系统已产生死锁则启动解锁程序以解除死锁。有关死锁检测及解锁技术可参阅相应的资料，这里不做进一步讨论。

5.6.2 数据恢复

把数据库从错误状态恢复到某一已知的正确状态的功能，称为数据库的恢复。数据库的故障可以分为事务内部的故障、系统故障、介质故障和计算机病毒造成的故障等。

1．事务内部的故障

1）可预期的

例如，把一笔金额从一个账户转给另一个账户：

```
Begin Transaction
Balance = Balance - Amount;
if(Balance < 0) Rollback;
else Balance1 = Balance1 + Amount;
Commit;
```

2）不可预期的

不可预期的事务内部的故障是指运算溢出、并发事务发生死锁、违反完整性约束等。

2．系统故障

系统故障包括硬件错误、操作系统错误、DBMS 代码错误和突然停电等。

数据恢复的基本原理就是冗余，建立冗余的方法有数据转储和登录日志文件等。可根据故障的不同类型，采用不同的恢复策略。

1）事务故障的恢复

事务故障的恢复由系统自动完成，对用户是透明的，步骤如下。

①反向扫描文件日志，查找该事务的更新操作。

②对该事务的更新操作执行逆操作。

③继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。

④如此处理下去，直至读到此事务的开始标记，事务故障恢复完成。

2）系统故障的恢复

系统故障的恢复在重新启动时自动完成，不需要用户干预。步骤如下。

①正向扫描日志文件，找出在故障发生前已经提交的事务，将其事务标识记入重做（Redo）队列。同时找出故障发生时尚未完成的事务，将其事务标识记入撤销（Undo）队列。

②对撤销队列中的各个事务进行撤销处理：反向扫描日志文件，对每个 Undo 事务的更新操作执行逆操作。

③对重做队列中的各个事务进行重做处理：正向扫描日志文件，对每个 Redo 事务重新执行日志文件登记的操作。

3）介质故障与病毒破坏的恢复

介质故障与病毒破坏的恢复步骤如下。

①装入最新的数据库后备副本，使数据库恢复到最近一次转储时的一致性状态。

②从故障点开始反向读日志文件，找出已提交事务标识并将其记入重做队列。

③从起始点开始正向阅读日志文件，根据重做队列中的记录，重做所有已完成事务，将数据库恢复至故障前某一时刻的一致状态。

4）具有检查点的恢复技术

检查点记录的内容可包括：

- 建立检查点时刻所有正在执行的事务清单。
- 这些事务最近一个日志记录的地址。

采用检查点的恢复步骤如下。

①从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录。

②由该检查点记录得到检查点建立时所有正在执行的事务清单队列（A）。

③建立重做队列（R）和撤销队列（U），把 A 队列放入 U 队列中，R 队列为空。

④从检查点开始正向扫描日志文件，若有新开始的事务 T1，则把 T1 放入 U 队列。若有提交的事务 T2，则把 T2 从 U 队列移到 R 队列；直至日志文件结束。

⑤对 U 队列的每个事务执行 Undo 操作，对 R 队列的每个事务执行 Redo 操作。

5.6.3 安全性

在数据库系统中大量数据集中存放，而且多用户共享，系统安全保护措施是否有效是数据库系统主要的性能指标之一。数据库安全模型如图 5-2 所示。

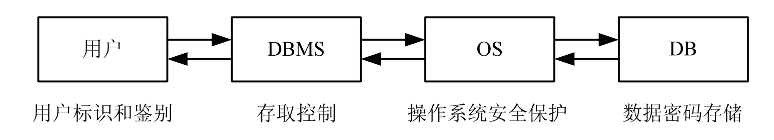


图 5-2　数据库安全模型

1．用户标识与鉴别

用户标识和鉴定是系统提供的最外层的安全保护措施。其方法是每次用户要进入系统时由系统提供一定的方式让用户标识自己的名字或身份，系统对用户身份进行鉴定核实后才提供系统使用权，常用的方法有如下几种。

- 用户名或用户标识号：在定义外模式时为每个用户提供一个用户代号存放在数据字典中。用户使用系统时，系统鉴别此用户是否是合法用户，若是，则可进入下一步的核实，否则不能使用系统。
- 口令：为了进一步核实用户，系统常常要求用户输入口令。为保密起见，用户在终端上输入的口令不显示在屏幕上，系统核对口令以鉴别用户身份。上述方法简单易行，但用户名、口令容易被人窃取，因此还可以用更可靠的方法。
- 随机数检验：用户根据预先约定好的计算公式求出一个数值作为动态口令送入计算机，当这个值与系统算出的结果一致时，才允许进入系统。

用户标识和鉴定可以重复多次。

2．存取控制

在数据库系统中，为了保证用户只能存取有权存取的数据，系统要求对每个用户定义存取权限。存取权限包括两方面的内容：一方面是要存取的数据对象；另一方面是对此数据对象进行操作的类型。对一个用户定义存取权限就是要定义这个用户可以在哪些数据对象上进行哪些类型的操作。在数据库系统中对存取权限的定义称为“授权”，这些授权定义经过编译后存放在数据库中。对于获得使用权又进一步发出存取数据库操作的用户，系统就根据事先定义好的存取权限进行合法权检查，若用户的操作超出了定义的权限，系统拒绝执行此操作，这就是存取控制。

授权编译程序和合法权检查机制一起组成了安全性子系统。

在非关系系统中，用户只能对数据进行操作，存取控制的数据对象也仅限于数据本身。而关系数据库系统中，DBA 可以把建立和修改基本表的权限授予用户，用户可利用这种权限来建立和修改基本表、索引、视图，因此，关系系统中存取控制的数据对象不仅有数据本身，还有模式、外模式、内模式等内容，如表 5-12 所示。

表 5-12　关系系统中的存取权限

	数 据 对 象	操 作 类 型
模式	模式	建立、修改、检索
	外模式	建立、修改、检索
	内模式	建立、修改、检索
数据	表	查找、插入、修改、删除
	属性列	查找、插入、修改、删除

关系数据语言 SQL 除数据定义和数据操作外，还提供了数据控制的功能，其授权和收回就是通过其提供的 GRANT 和 REVOKE 语句来实现的。

3．视图机制

视图机制可以将要保密的数据对无权存取这些数据的用户隐藏起来，这样就自动地提供了对数据的安全保护。

4．审计

审计是现代计算机系统中必不可少的功能之一，其主要任务是对用户（包括应用程序）使用系统资源（包括软/硬件和数据）的情况进行记录和审查，一旦发现问题，审计人员通过审计跟踪，可望找出原因，追查责任，防止类似问题再度发生。因此，审计往往作为保证数据库安全的一种补救措施。

数据库系统中的审计工作包括如下几种。

- 设备安全审计。主要审查关于系统资源的安全策略、各种安全保护措施及故障恢复计划等。
- 操作审计。对系统的各种操作（特别是一些敏感操作）进行记录、分析。记录内容包括操作的种类、所属事务、所属进程、用户、终端（或客户机）、操作时间、审计日期等。
- 应用审计。审计建于数据库之上的整个应用系统的功能、控制逻辑、数据流是否正确。
- 攻击审计。对已发生的攻击性操作及危害系统安全的事件（或企图）进行检测和审计。

上述各种审计所用技术大致可分为如下 3 类。

1）*静态分析系统技术*

审计者通过查阅各种系统资源（软/硬件、数据）的说明性文件，如软件的设计说明书、流程图等来了解整个系统，甚至定位出一些易被攻击的薄弱环节。

2）*运行验证技术*

运行验证的目的是保证系统控制逻辑正确，各类事务能有效执行。该技术一般又细分为实际运行测试和性能测试两种。实现时，审计者既可根据审计需要，选择系统中一个实际事务作为样板进行审计跟踪，又可生成专门的测试用例，通过将测试用例在系统运行的实际结果与期望结果进行比较来评价系统；还可设计一个专门仿真系统的程序，让仿真系统与实际系统并行工作，比较它们的结果来评测系统。

3）*运行结果验证技术*

这种技术注意力放在运行结果——数据上。它主要涉及审计数据选择和收集、数据分析两类问题。常用的审计数据选择和收集的办法有：

- 在应用程序中插入一个审计数据收集模块。
- 设置专门的审计跟踪事务。
- 兼用系统的日志库。
- 使用由随机抽取记录组成的专用审计库。

一旦获得审计数据后，审计者可以检查各类控制信息、完整性约束等内容，以达到各种审计目的。

5．数据加密

对于那些保密程度极高的数据（如用户标识、绝密信息等）和在网络传输过程中可能被盗窃的数据除采用上述安全保护措施外，一般还需采用数据加密技术，以密文形式保存和传输，保证只有那些知道密钥的用户可以访问。数据加密是防止数据库中的数据在存储和传输中失密的有效手段。有关加密技术的详细内容可参考相关章节，在此不再详细叙述。

还有一种统计数据库安全性，举例说明如下。

例 1

（1）本公司共有多少女高级程序员？

（2）本公司女高级程序员的工资总额是多少？

问题：

如果（1）中查询的结果为“1”，那么（2）中查询的结果就是该高级程序员的工资。

例 2　设某用户 A 的工资是 Z，他想知道用户 B 的工资。

（1）用户 A 和其他 *N* 个程序员的工资总额是多少？

（2）用户 B 和其他 *N* 个程序员的工资总额是多少？

问题：

如果（1）中查询的结果为 X，（2）中查询的结果是 Y，那么 B 的工资为 Y－X+Z。

5.6.4　完整性

1．完整性约束条件

保证数据库中的数据完整性的方法之一是设置完整性检查，即对数据库中的数据设置一些约束条件，这是数据的语义体现。数据的完整性约束条件一般在数据模式中给出，并在运行时检查，当不满足条件时立即向用户通报以便采取措施，如表 5-13 所示。

完整性约束条件一般指的是对数据库中数据本身的某些语法、语义限制，数据间的逻辑约束及数据变化时应遵守的规则等。所有这些约束条件一般均以谓词逻辑形式表示，即以具有真假值的原子公式及命题连接词（并且、或者、否定）所组成的逻辑公式表示。完整性约束条件的作用对象可以是关系、元组、列三种。

数据库中数据的语法、语义限制与数据间的逻辑约束称为静态约束。它反映了数据及数据间的固有的逻辑特性，是最重要的一类完整性约束。静态约束包括静态列级约束（对数据类型的约束、对数据格式的约束、对取值范围或取值集合的约束、对空值的约束、其他约束）、静态元组约束、静态关系约束（实体完整性约束、参照完整性约束、函数依赖约束、统计约束）。

数据库中的数据变化应遵守的规则称为数据动态约束，它反映了数据库状态变迁的约束。动态约束包括动态列级约束（修改列定义时的约束、修改列值时的约束）、动态元组约束、动态关系约束。

表 5-13　完整性约束条件

粒度 / 状态	数 据 对 象	元 组 级	操 作 类 型
静态	列定义 ● 类型 ● 格式 ● 值域 ● 空值	元组值应 满足的条件	实体完整性约束 参照完整性约束 函数依赖约束 统计约束
动态	改变列定义或列值	元组新旧值之间应满足的约束条件	关系新旧状态间应满足的约束条件

2．完整性控制

1）完整性控制机制应该具有的功能

- 定义功能：提供定义完整性约束条件的机制。
- 检查功能：检查用户发出的操作请求是否违背了完整性约束条件。

如果发现用户的操作请求违背了约束条件，则采取一定的动作来保证数据的完整性。

如果在一条语句执行完后立即检查，则称为立即执行约束；如果在整个事务执行结束后再进行检查，则称为延迟执行约束。

2）完整性规则的五元组（D，O，A，C，P）

- D：约束作用的数据对象。
- O：触发完整性检查的数据库操作。
- A：数据对象必须满足的断言或语义约束。
- C：选择A作用的数据对象值的谓词。
- P：违反完整性规则时触发的过程。

如学号不能为空可表示为如下。

- D：约束作用的数据对象为Sno属性。
- O：插入或修改元组时。
- A：Sno定义为Not Null。
- C：无。
- P：拒绝执行该操作。

3）参照完整性

- 外码能否接受空值问题根据实际应用决定。
- 在被参照关系中删除元组的问题。
- 级联删除（Cascades）：将参照关系中所有外码值与被参照关系中要删除元组的主码值相同的元组一起删除。如果参照关系同时又是另一个关系的被参照关系，则这种删除操作会继续级联下去。
- 受限删除（Restrict 默认）：仅当参照关系中没有任何元组的外码值与被参照关系中要删除元组的主码值相同时，系统才可以执行删除操作，否则拒绝执行删除操作。
- 置空删除（Set Null）：删除被参照关系的元组时，并将参照关系中相应元组的外码值置为空值。
- 在参照关系中插入元组的问题。
- 受限插入：仅当被参照关系中存在相应的元组时，其主码值与参照关系插入元组的外码值相同时，系统才执行插入操作，否则拒绝此操作。
- 递归插入：首先向被参照关系中插入相应的元组，其主码值等于参照关系插入元组的外码值，然后向参照关系插入元组。
- 修改关系中主码的问题。
- 不允许修改主码。
- 允许修改主码。

4）触发器

触发器（Trigger）是在关系数据库管理系统中应用得比较多的一种完整性保护措施。触发器的功能一般比完整性约束要强得多，一般而言在完整性约束功能中，当系统检查出数据中有违反完整性约束条件时，则仅给出必要提示以通知用户，仅此而已。而触发器的功能则不仅仅起提示作用，它还会引起系统内自动进行某些操作以消除违反完整性约束条件所引起的负面影响。

所谓触发器，其抽象的含义即是一个事件的发生必然触发（或导致）另外一些事件的发生，其中前面的事件称为触发事件，后面的事件称为结果事件。触发事件一般即为完整性约束条件的否定。而结果事件即为一组操作用以消除触发事件所引起的不良影响。在目前数据库中事件一般表示为数据的插入、修改、删除等操作。

触发器除有完整性保护功能外，还有安全性保护功能。

5.7 数据仓库与数据挖掘

本节将介绍数据仓库与数据挖掘。

5.7.1 数据仓库的概念

目前，数据仓库一词尚没有一个统一的定义，著名的数据仓库专家 W.H.Inmon 在其著作 *Building the Data Warehouse* 中给予了如下描述：数据仓库（Data Warehouse）是一个面向主题的、集成的、相对稳定的且随时间变化的数据集合，用于支持管理决策。

1. 面向主题

操作型数据库的数据组织面向事务处理任务（面向应用），各个业务系统之间各自分离，而数据仓库中的数据是按照一定的主题域进行组织的。主题是一个抽象的概念，是指用户使用数据仓库进行决策时所关心的重点方面，一个主题通常与多个操作型信息系统相关。例如，一个保险公司所进行的事务处理（应用问题）可能包括汽车保险、人寿保险、健康保险和意外保险等，而公司的主要主题范围可能是顾客、保险单、保险费和索赔等。

2. 集成的

在数据仓库的所有特性中，这是最重要的。面向事务处理的操作型数据库通常与某些特定的应用相关，数据库之间相互独立，并且往往是异构的。而数据仓库中的数据是在对原有分散的数据库数据抽取、清理的基础上经过系统加工、汇总和整理得到的，必须消除源数据中的不一致性，以保证数据仓库内的信息是关于整个企业的一致的全局信息。

3. 相对稳定的

操作型数据库中的数据通常实时更新，数据根据需要及时发生变化。数据仓库的数据主要供企业决策分析之用，所涉及的数据操作主要是数据查询，一旦某个数据进入数据仓库后，一般情况下将被长期保留，也就是数据仓库中一般有大量的查询操作，但修改和删除操作很少，通常只需要定期的加载、刷新。

4．随时间变化

操作型数据库主要关心当前某一个时间段内的数据，而数据仓库中的数据通常包含历史信息，系统记录了企业从过去某一时点（如开始应用数据仓库的时点）到目前的各个阶段的信息，通过这些信息，可以对企业的发展历程和未来趋势做出定量分析和预测。

数据仓库反映历史变化的属性主要表现在如下方面。

- 数据仓库中的数据时间期限要远远长于传统操作型数据系统中的数据时间期限。传统操作型数据系统中的数据时间期限可能为数十天或数个月，数据仓库中的数据时间期限往往为数年甚至几十年。
- 传统操作型数据系统中的数据含有“当前值”的数据，这些数据在访问时是有效的，当然数据的当前值也能被更新，但数据仓库中的数据仅仅是一系列某一时刻（可能是传统操作型数据系统）生成的复杂的快照。
- 传统操作型数据系统中可能包含也可能不包含时间元素，如年、月、日、时、分、秒等，而数据仓库中一定会包含时间元素。

数据仓库虽然是从传统数据库系统发展而来的，但两者还是存在着诸多差异，例如，从数据存储的内容看，数据库只存放当前值，而数据仓库则存放历史值；数据库数据的目标是面向业务操作人员的，为业务处理人员提供数据处理的支持，而数据仓库则是面向中高层管理人员的，为其提供决策支持等。表5-14详细说明了数据仓库与传统数据库的区别。

表 5-14　数据仓库与传统数据库的比较

比 较 项 目	传统数据库	数 据 仓 库
数据内容	当前值	历史的、归档的、归纳的、计算的数据（处理过的）
数据目标	面向业务操作程序、重复操作	面向主体域，分析应用
数据特性	动态变化、更新	静态、不能直接更新，只能定时添加、更新
数据结构	高度结构化、复杂，适合操作计算	简单、适合分析
使用频率	高	低
数据访问量	每个事务一般只访问少量记录	每个事务一般访问大量记录
对响应时间的要求	计时单位小，如秒	计时单位相对较大，除了秒，还有分钟、小时

5.7.2　数据仓库的结构

1．数据仓库的概念结构

从数据仓库的概念结构看，一般来说，数据仓库系统要包含数据源、数据准备区、数据仓库数据库、数据集市/知识挖掘库，以及各种管理工具和应用工具，如图 5-3 所示。数据仓库建立之后，首先要从数据源中抽取相关的数据到数据准备区，在数据准备区中经过净化处理后再加载到数据仓库数据库，最后根据用户的需求将数据导入数据集市和知识挖掘库中。当用户使用数据仓库时，可以利用包括 OLAP 在内的多种数据仓库应用工具向数据集市/知识挖掘库或数据仓库进行决策查询分析或知识挖掘。数据仓库的创建、应用可以利用各种数据仓库管理工具辅助完成。

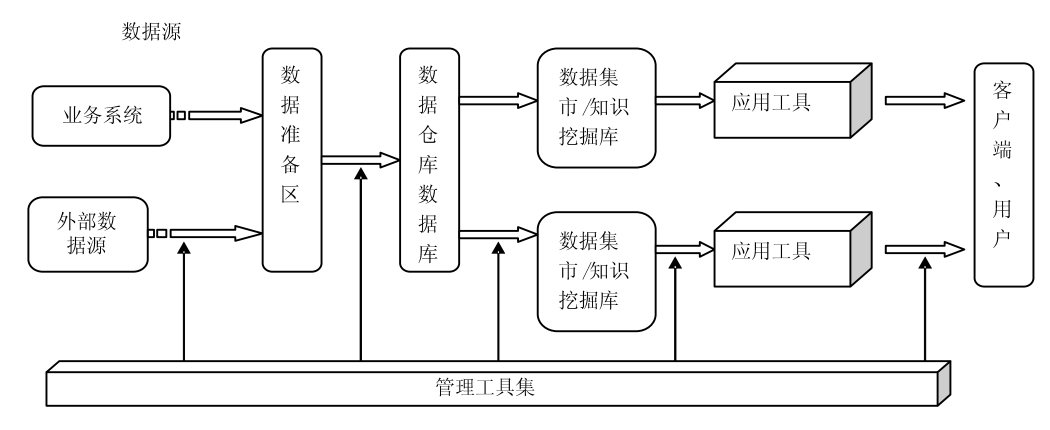


图 5-3　数据仓库的概念结构

2．数据仓库的参考框架

数据仓库的参考框架由数据仓库的基本功能层、数据仓库的管理层和数据仓库的环境支持层组成，如图 5-4 所示。

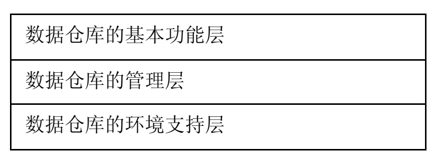


图 5-4　数据仓库的框架结构

1）数据仓库的基本功能层

数据仓库的基本功能层部分包含数据源、数据准备区、数据仓库结构、数据集市或知识挖掘库，以及存取和使用部分。本层的功能是从数据源抽取数据，对所抽取的数据进行筛选、清理，将处理过的数据导入或者加载到数据仓库中，根据用户的需求设立数据集市，完成数据仓库的复杂查询、决策分析和知识的挖掘等。

2）数据仓库的管理层

数据仓库的正常运行除需要数据仓库功能层提供的基本功能外，还需要对这些基本功能进行管理与支持的结构框架。数据仓库管理层由数据仓库的数据管理和数据仓库的元数据管理组成。

数据仓库的数据管理层包含数据抽取、新数据需求与查询管理，数据加载、存储、刷新和更新系统，安全性与用户授权管理系统，以及数据归档、恢复及净化系统四部分。

3）数据仓库的环境支持层

数据仓库的环境支持层由数据仓库数据传输层和数据仓库基础层组成。数据仓库中不同结构之间的数据传输需要数据仓库的传输层来完成。

数据仓库的传输层包含数据传输和传送网络，客户/服务器代理和中间件，复制系统，以及数据传输层的安全保障系统。

3．数据仓库的体系结构

大众观点的数据仓库的体系结构如图 5-5 所示。

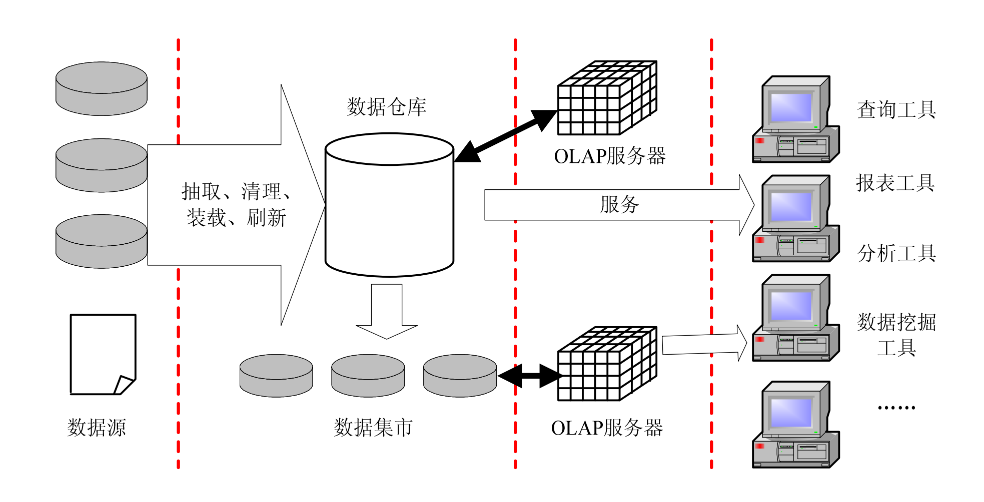


图 5-5　数据仓库体系结构

1）数据源

数据源是数据仓库系统的基础，是整个系统的数据源泉。通常包括企业内部信息和外部信息。内部信息包括存放于 RDBMS 中的各种业务处理数据和各类文档数据。外部信息包括各类法律法规、市场信息和竞争对手的信息等。

2）数据的存储与管理

它是整个数据仓库系统的核心。数据仓库的真正关键是数据的存储和管理。数据仓库的组织管理方式决定了它有别于传统数据库，同时也决定了其对外部数据的表现形式。要决定采用什么产品和技术来建立数据仓库的核心，则需要从数据仓库的技术特点着手分析。针对现有各业务系统的数据，进行抽取、清理，并有效集成，按照主题进行组织。数据仓库按照数据的覆盖范围可以分为企业级数据仓库和部门级数据仓库（通常称为数据集市）。

3）OLAP 服务器

OLAP 服务器对分析需要的数据进行有效集成，按多维模型予以组织，以便进行多角度、多层次的分析，并发现趋势。其具体实现可以分为：ROLAP、MOLAP 和 HOLAP。ROLAP 基本数据和聚合数据均存放于 RDBMS 中；MOLAP 基本数据和聚合数据均存放于多维数据库中；HOLAP 基本数据存放于 RDBMS 中，聚合数据存放于多维数据库中。

4）前端工具

前端工具主要包括各种报表工具、查询工具、数据分析工具、数据挖掘工具，以及各种基于数据仓库或数据集市的应用开发工具。其中数据分析工具主要针对 OLAP 服务器，报表工具、数据挖掘工具主要针对数据仓库。

5.7.3　数据挖掘技术概述

随着数据库技术的迅速发展及数据库管理系统的广泛应用，人们积累的数据越来越多。激增的数据背后隐藏着许多重要的信息，人们希望能够对其进行更高层次的分析，以便更好地利用这些数据。目前的数据库系统可以高效地实现数据的录入、查询、统计等功能，但无法发现数据中存在的关系和规则，无法根据现有的数据预测未来的发展趋势。缺乏挖掘数据背后隐藏的知识的手段，导致了“数据爆炸但知识贫乏”的现象。

数据挖掘（Data Mining）技术是人们长期对数据库技术进行研究和开发的结果。起初各种商业数据是存储在计算机的数据库中的，然后发展到可对数据库进行查询和访问，进而发展到对数据库的即时遍历。数据挖掘使数据库技术进入了一个更高级的阶段，它不仅能对过去的数据进行查询和遍历，并且能够找出过去数据之间的潜在联系，从而促进信息的传递。现在数据挖掘技术在商业应用中已经可以马上投入使用，因为对这种技术进行支持的三种基础技术已经发展成熟，它们是海量数据搜集、强大的多处理器计算机和数据挖掘算法。

从技术上来看，数据挖掘就是从大量的、不完全的、有噪声的、模糊的、随机的实际应用数据中，提取隐含在其中的、人们事先不知道的但又是潜在有用的信息和知识的过程。这个定义包括好几层含义：数据源必须是真实的、大量的、含噪声的；发现的是用户感兴趣的知识；发现的知识要可接受、可理解、可运用；并不要求发现放之四海而皆准的知识，仅支持特定的发现问题。

还有很多和这一术语相近似的术语，如从数据库中发现知识（KDD）、数据分析、数据融合（Data Fusion）及决策支持等。

何为知识？从广义上理解，数据、信息也是知识的表现形式，但是人们更把概念、规则、模式、规律和约束等看作知识。原始数据可以是结构化的，如关系数据库中的数据；也可以是半结构化的，如文本、图形和图像数据；甚至是分布在网络上的异构型数据。发现知识的方法可以是数学的，也可以是非数学的；可以是演绎的，也可以是归纳的。发现的知识可以被用于信息管理，查询优化，决策支持和过程控制等，还可以用于数据自身的维护。因此，数据挖掘是一门交叉学科，它把人们对数据的应用从低层次的简单查询，提升到从数据中挖掘知识，提供决策支持。在这种需求牵引下，汇聚了不同领域的研究者，尤其是数据库技术、人工智能技术、数理统计、可视化技术、并行计算等方面的学者和工程技术人员，投身到数据挖掘这一新兴的研究领域，形成新的技术热点。

从商业角度来看，数据挖掘是一种新的商业信息处理技术，其主要特点是对商业数据库中的大量业务数据进行抽取、转换、分析和其他模型化处理，从中提取辅助商业决策的关键性数据。

简而言之，数据挖掘其实是一类深层次的数据分析方法。数据分析本身已经有很多年的历史，只不过在过去数据收集和分析的目的是用于科学研究，另外，由于当时计算能力的限制，对大数据量进行分析的复杂数据分析方法受到很大限制。现在，由于各行业业务自动化的实现，商业领域产生了大量的业务数据，这些数据不再是为了分析的目的而收集的，而是由于纯机会的（Opportunistic）商业运作而产生的。分析这些数据也不再是单纯为了研究的需要，更主要的是为商业决策提供真正有价值的信息，进而获得利润。但所有企业面临的一个共同问题是：企业数据量非常大，而其中真正有价值的信息却很少，因此从大量的数据中经过深层分析，获得有利于商业运作、提高竞争力的信息，就像从矿石中淘金一样，数据挖掘也因此而得名。

因此，数据挖掘可以描述为：按企业既定业务目标，对大量的企业数据进行探索和分析，揭示隐藏的、未知的或验证已知的规律性，并进一步将其模型化的先进有效的方法。

数据挖掘与传统的数据分析（如查询、报表、联机应用分析）的本质区别是数据挖掘是在没有明确假设的前提下去挖掘信息、发现知识。数据挖掘所得到的信息应具有先知、有效和可实用 3 个特征。

先前未知的信息是指该信息是预先未曾预料到的，即数据挖掘是要发现那些不能靠直觉发现的信息或知识，甚至是违背直觉的信息或知识，挖掘出的信息越是出乎意料，就可能越有价值。在商业应用中最典型的例子就是一家连锁店通过数据挖掘发现了小孩尿布和啤酒之间有着惊人的联系。

特别要指出的是，数据挖掘技术从一开始就是面向应用的。它不仅要面向特定数据库进行简单检索查询调用，而且要对这些数据进行微观、中观乃至宏观的统计、分析、综合和推理，以指导实际问题的求解，企图发现事件间的相互关联，甚至利用已有的数据对未来的活动进行预测。例如，加拿大 BC 省电话公司要求加拿大 SimonFraser 大学 KDD 研究组，根据其拥有十多年的客户数据，总结、分析并提出新的电话收费和管理办法，制订既有利于公司又有利于客户的优惠政策。这样一来，就把人们对数据的应用，从低层次的末端查询操作，提高到为各级经营决策者提供决策支持。这种需求驱动力比数据库查询更为强大。

5.7.4 数据挖掘的功能

数据挖掘通过预测未来趋势及行为，做出前摄的、基于知识的决策。数据挖掘的目标是从数据库中发现隐含的、有意义的知识，主要有如下 5 类功能：

1．自动预测趋势和行为

数据挖掘自动在大型数据库中寻找预测性信息，以往需要进行大量手工分析的问题如今可以迅速直接由数据本身得出结论。一个典型的例子是市场预测问题，数据挖掘使用过去有关促销的数据来寻找未来投资中回报最大的用户，其他可预测的问题包括预报破产及认定对指定事件最可能做出反应的群体。

2．关联分析

数据关联是数据库中存在的一类重要的可被发现的知识。若两个或多个变量的取值之间存在某种规律性，就称为关联。关联可分为简单关联、时序关联、因果关联。关联分析的目的是找出数据库中隐藏的关联网。有时并不知道数据库中数据的关联函数，即使知道也是不确定的，因此关联分析生成的规则带有可信度。

3．聚类

数据库中的记录可被划分为一系列有意义的子集，即聚类。聚类增强了人们对客观现实的认识，是概念描述和偏差分析的先决条件。聚类技术主要包括传统的模式识别方法和数学分类学。20 世纪 80 年代初，Mchalski 提出了概念聚类技术及其要点，即在划分对象时不仅考虑对象之间的距离，还要求划分出的类具有某种内涵描述，从而避免了传统技术的某些片面性。

4．概念描述

概念描述就是对某类对象的内涵进行描述，并概括这类对象的有关特征。概念描述分为特征性描述和区别性描述，前者描述某类对象的共同特征，后者描述不同类对象之间的区别。生成一个类的特征性描述只涉及该类对象中所有对象的共性。生成区别性描述的方法很多，如决策树方法、遗传算法等。

5．偏差检测

数据库中的数据常有一些异常记录，从数据库中检测这些偏差很有意义。偏差包括很多潜在的知识，如分类中的反常实例、不满足规则的特例、观测结果与模型预测值的偏差、量值随时间的变化等。偏差检测的基本方法是，寻找观测结果与参照值之间有意义的差别。

5.7.5 数据挖掘常用技术

常用的数据挖掘技术包括关联分析、序列分析、分类、预测、聚类分析，以及时间序列分析等。

1．关联分析

关联分析主要用于发现不同事件之间的关联性，即一个事件发生的同时，另一个事件也经常发生。关联分析的重点在于快速发现那些有实用价值的关联发生的事件。其主要依据是事件发生的概率和条件概率应该符合一定的统计意义。

对于结构化的数据，以客户的购买习惯数据为例，利用关联分析，可以发现客户的关联购买需要。例如，一个开设储蓄账户的客户很可能同时进行债券交易和股票交易，购买纸尿裤的男顾客经常同时购买啤酒等。利用这种知识可以采取积极的营销策略，扩展客户购买的产品范围，吸引更多的客户。通过调整商品的布局便于顾客买到经常同时购买的商品，或者通过降低一种商品的价格来促进另一种商品的销售等。

对于非结构化的数据，以空间数据为例，利用关联分析，可以发现地理位置的关联性。例如，85%的靠近高速公路的大城镇与水相邻，或者发现通常与高尔夫球场相邻的对象等。

2．序列分析

序列分析技术主要用于发现一定时间间隔内接连发生的事件。这些事件构成一个序列，发现的序列应该具有普遍意义，其依据除统计上的概率之外，还要加上时间的约束。

3．分类分析

分类分析通过分析具有类别的样本的特点，得到决定样本属于各种类别的规则或方法。利用这些规则和方法对未知类别的样本分类时应该具有一定的准确度。其主要方法有基于统计学的贝叶斯方法、神经网络方法、决策树方法等。

利用分类技术，可以根据顾客的消费水平和基本特征对顾客进行分类，找出对商家有较大利益贡献的重要客户的特征，通过对其进行个性化服务，提高他们的忠诚度。

利用分类技术，可以将大量的半结构化的文本数据，如 Web 页面、电子邮件等进行分类。可以将图片进行分类，例如，根据已有图片的特点和类别，可以判定一幅图片属于何种类型的规则。对于空间数据，也可以进行分类分析，例如，可以根据房屋的地理位置决定房屋的档次。

4．聚类分析

聚类分析是根据物以类聚的原理，将本身没有类别的样本聚集成不同的组，并且对每一个这样的组进行描述的过程。其主要依据是聚到同一个组中的样本应该彼此相似，而属于不同组的样本应该足够不相似。

仍以客户关系管理为例，利用聚类技术，根据客户的个人特征及消费数据，可以将客户群体进行细分。例如，可以得到这样的一个消费群体：女性占 91%，全部无子女、年龄在 31～40 岁占 70%，高消费级别的占 64%，买过针织品的占 91%，买过厨房用品的占 89%，买过园艺用品的占 79%。针对不同的客户群，可以实施不同的营销和服务方式，从而提高客户的满意度。

对于空间数据，根据地理位置及障碍物的存在情况可以自动进行区域划分。例如，根据分布在不同地理位置的 ATM 机的情况将居民进行区域划分，根据这一信息，可以有效地

进行 ATM 机的设置规划，避免浪费，同时也避免失掉每一个商机。

对于文本数据，利用聚类技术可以根据文档的内容自动划分类别，从而便于文本的检索。

5．预测

预测与分类类似，但预测是根据样本的已知特征估算某个连续类型的变量的取值过程，而分类则只是用于判别样本所属的离散类别而已。预测常用的技术是回归分析。

6．时间序列分析

时间序列分析的是随时间而变化的事件序列，目的是预测未来发展趋势，或者寻找相似发展模式或者发现周期性发展规律。

5.7.6 数据挖掘的流程

数据挖掘是指一个完整的过程，该过程从大型数据库中挖掘先前未知的、有效的、可实用的信息，并利用这些信息做出决策或丰富知识。

数据挖掘环境示意图如图 5-6 所示。

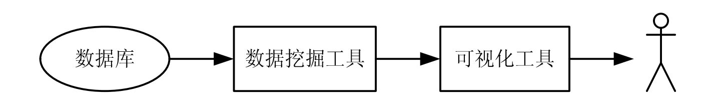


图 5-6 数据挖掘环境示意图

数据挖掘的流程大致如下。

1．问题定义

在开始数据挖掘之前最先的也是最重要的要求就是熟悉背景知识，弄清用户的需求。缺少了背景知识，就不能明确定义要解决的问题，就不能为挖掘准备优质的数据，也很难正确地解释所得到的结果。要想充分发挥数据挖掘的价值，必须对目标有一个清晰明确的定义，即决定到底想干什么。

2．建立数据挖掘库

要进行数据挖掘必须收集要挖掘的数据资源。一般建议把要挖掘的数据都收集到一个数据库中，而不是采用原有的数据库或数据仓库。这是因为大部分情况下需要修改要挖掘的数据，而且还会遇到采用外部数据的情况；另外，数据挖掘还要对数据进行各种纷繁复杂的统计分析，而数据仓库可能不支持这些数据结构。

3．分析数据

分析数据就是通常所进行的对数据深入调查的过程。从数据集中找出规律和趋势，用聚类分析区分类别，最终要达到的目的就是搞清楚多因素相互影响的、十分复杂的关系，发现因素之间的相关性。

4．调整数据

通过上述步骤的操作，对数据的状态和趋势有了进一步的了解，这时要尽可能对问题解决的要求能进一步明确化，进一步量化。针对问题的需求对数据进行增删，按照对整个数据挖掘过程的新认识组合或生成一个新的变量，以体现对状态的有效描述。

5．模型化

在问题进一步明确，数据结构和内容进一步调整的基础上，就可以建立形成知识的模型。这一步是数据挖掘的核心环节，一般运用神经网络、决策树、数理统计、时间序列分析等方法来建立模型。

6．评价和解释

上面得到的模式模型，有可能是没有实际意义或没有实用价值的，也有可能是其不能准确反映数据的真实意义，甚至在某些情况下是与事实相反的，因此需要评估，确定哪些是有效的、有用的模式。评估的一种办法是直接使用原先建立的挖掘数据库中的数据来进行检验，另一种办法是另找一批数据并对其进行检验，再一种办法就是在实际运行的环境中取出新鲜数据进行检验。

数据挖掘过程的分步实现，不同的步骤需要不同专长的人员，他们大体可以分为三类。

- 业务分析人员：要求精通业务，能够解释业务对象，并根据各业务对象确定出用于数据定义和挖掘算法的业务需求。
- 数据分析人员：精通数据分析技术，并对统计学有较熟练的掌握，有能力把业务需求转化为数据挖掘的各步操作，并为每步操作选择合适的技术。
- 数据管理人员：精通数据管理技术，并从数据库或数据仓库中收集数据。

由此可见，数据挖掘是一个多种专家合作的过程，也是一个在资金上和技术上高投入的过程。这一过程要反复进行，在反复过程中，不断地趋近事物的本质，不断地优选问题的解决方案。

5.8 分布式数据库

1．分布式数据库系统的定义与特点

分布式数据库是由一组数据组成的，这组数据分布在计算机网络的不同计算机上，网络中的每个结点具有独立处理的能力（称为场地自治），它可以执行局部应用，同时，每个结点也能通过网络通信子系统执行全局应用。

分布式数据库系统是在集中式数据库系统技术的基础上发展起来的，具有如下特点。

- 数据独立性：在分布式数据库系统中，数据独立性这一特性更加重要，并具有更多的内容。除数据的逻辑独立性与物理独立性外，还有数据分布独立性，也称为“分布透明性”。
- 集中与自治共享结合的控制结构：各局部的DBMS可以独立地管理局部数据库，具有自治的功能。同时，系统又设有集中控制机制，协调各局部DBMS的工作，执行全局应用。
- 适当增加数据冗余度：在不同的场地存储同一数据的多个副本，这样可以提高系统的可靠性、可用性，同时也能提高系统性能。
- 全局的一致性、可串行性和可恢复性。

分布式数据库系统的目标，主要包括技术和组织两方面的目标。

- 适应部门分布的组织结构，降低费用。
- 提高系统的可靠性和可用性。

- 充分利用数据库资源，提高现有集中式数据库的利用率。
- 逐步扩展处理能力和系统规模。

2．分布式数据存储

分布式数据存储可以从数据分配和数据分片两个角度来考查。

数据分配是指数据在计算机网络各场地上的分配策略，包括如下内容。

- 集中式：所有数据均安排在同一个场地上。
- 分割式：所有数据只有一份，分别被安置在若干个场地。
- 全复制式：数据在每个场地重复存储。
- 混合式：数据库分成若干可相交的子集，每一子集安置在一个或多个场地上，但是每一场地未必保存全部数据。

对于上述分配策略，有 4 个评估因素。

①存储代价；②可靠性；③检索代价；④更新代价。

存储代价和可靠性是一对矛盾的因素；检索代价和更新代价也是一对矛盾的因素。

数据分片是指数据存放单位不是全部关系，而是关系的一个片段，也就是关系的一部分，包括如下内容。

- 水平分片：按一定的条件把全局关系的所有元组划分成若干不相交的子集，每个子集为关系的一个片段。
- 垂直分片：把一个全局关系的属性集分成若干子集，并在这些子集上做投影运算，每个投影为垂直分片。
- 混合型分片：将水平分片与垂直分片方式综合使用，则为混合型分片。

数据分片应遵循的准则如下。

- 完备性条件：必须把全局关系的所有数据映射到各个片段中，绝不允许发生属于全局关系的某个数据不属于任何一个片段的情况。
- 重构条件：划分所采用的方法必须确保能够由各个片段重建全局关系。
- 不相交条件：要求一个全局关系被划分后得到的各个数据片段互相不重叠。

3．分布式数据库系统的体系结构

分布式 DBS 的体系结构分为 4 级：全局外模式、全局概念模式、分片模式和分布模式。

- 全局外模式：它们是全局应用的用户视图，是全局概念模式的子集。
- 全局概念模式：全局概念模式定义了分布式数据库中所有数据的逻辑结构。
- 分片模式：分片模式定义片段及定义全局关系与片段之间的映像。这种映像是一对多的，即每个片段来自一个全局关系，而一个全局关系可分成多个片段。
- 分布模式：片段是全局关系的逻辑部分，一个片段在物理上可以分配到网络的不同结点上。分布模式根据数据分配策略的选择定义片段的存放场地。

分布式 DBS 的分层体系结构有如下 3 个特征：

- 数据分片和数据分配概念的分离，形成了“数据分布独立性”概念。
- 数据冗余的显式控制。
- 局部DBMS的独立性。

4．分布透明性

分布透明性指用户不必关心数据的逻辑分片，不必关心数据物理位置分配的细节， 也不必关心各个场地上数据库数据模型。分布透明性可归入物理独立性的范围。

分布透明性包括 3 个层次：分片透明性、位置透明性和局部数据模型透明性。

5．分布式数据库管理系统的功能及组成

分布式数据管理系统的主要功能如下：

- 接受用户请求，并判定把它送到哪里，或必须访问哪些计算机才能满足该请求。
- 访问网络数据字典，或者至少了解如何请求和使用其中的信息。
- 如果目标数据存储于系统的多个计算机上，就必须进行分布式处理。
- 通信接口功能，在用户、局部DBMS和其他计算机的DBMS之间进行协调。
- 在一个异构型分布式处理环境中，还需提供数据和进程移植的支持。这里的异构型是指各个场地的硬件、软件之间存在一定差别。

D-DBMS 由如下 4 个部分组成：

- LDBMS。局部场地上的数据库管理系统的功能是建立和管理局部数据库，提供场地自治能力、执行局部应用及全局查询的子查询。
- GDBMS。全局数据库管理系统主要功能是提供分布透明性，协调全局事务的执行，协调各局部DBMS以完成全局应用，保证数据库的全局一致性，执行并发控制，实现更新同步，提供全局恢复功能。
- 全局数据字典存放全局概念模式、分片模式、分布模式的定义，以及各模式之间映像的定义，存放有关用户存取权限的定义，以保证全局用户的合法权限和数据库的安全性，存放数据完整性约束条件的定义，其功能与集中式数据库的数据字典类似。
- 通信管理在分布数据库各场地之间传送消息和数据，完成通信功能。

6．分布式数据库系统要解决的问题

在集中式系统中，主要目标是减少对磁盘的访问次数。对于分布式系统，压倒一切的性能目标是使通过网络传送信息的次数和数据量最小。

第 6 章

多媒体技术及其应用

根据考试大纲的规定，本章要求考生掌握如下知识：

- 多媒体系统基础知识。
- 简单图形的绘制，图像文件的处理方法。
- 音频和视频信息的应用。
- 多媒体应用开发过程。

但从历年考试试题来看，主要集中在音频、视频、图形和图像等方面。考生在复习时应掌握基本概念，熟悉有关的多媒体文件容量、量化方面的计算。

在多媒体应用开发过程方面，考生要注意多媒体应用系统具有如下特点：

- 增强了计算机的友好性；
- 涉及技术领域广、技术层次高；
- 多媒体技术的标准化；
- 多媒体技术的集成化和工具化。

多媒体应用系统开发组需要应用系统组长、多媒体设计师、音频专家、视频专家、写作专家、多媒体程序员等人员，其具体开发过程与非多媒体项目类似，读者可参考本书有关软件工程知识的章节。

6.1 多媒体技术基本概念

多媒体主要是指文字、声音和图像等多种表达信息的形式和媒体，它强调多媒体信息的综合和集成处理。多媒体技术依赖于计算机的数字化和交互处理能力，它的关键是信息压缩技术和光盘存储技术。

1. 亮度、色调和饱和度

视觉上的彩色可用亮度、色调和饱和度来描述，任意一种彩色光都是这 3 个特征的综合效果。

亮度是光作用于人眼时所引起的明亮程度的感觉，它与被观察物体的发光强度有关；由于其强度不同，看起来可能亮一些或暗一些。对于同一物体照射的光越强，反射光也越强，感觉越亮，对于不同物体在相同照射情况下，反射性越强者看起来越亮。显然，如果彩色光的强度降至使人看不清，在亮度等级上它应与黑色对应；同样如果其强度变得很大，那么亮度等级应与白色对应。此外，亮度感还与人类视觉系统的视敏功能有关，即使强度

相同，颜色不同的光进入视觉系统，也可能会产生不同的亮度。

色调是当人眼看到一种或多种波长的光时所产生的彩色感觉，它反映颜色的种类，是决定颜色的基本特性，如红色、绿色等都是指色调。不透明物体的色调是指该物体在日光照射下，所反射的各光谱成分作用于人眼的综合效果；透明物体的色调则是透过该物体的光谱综合作用的效果。

饱和度是指颜色的纯度，即掺入白光的程度，或者说是指颜色的深浅程度。对于同一色调的彩色光，饱和度越深，颜色越鲜明，或者说越纯。例如，当红色加进白光之后冲淡为粉红色，其基本色调还是红色，但饱和度降低；换句话说，淡色的饱和度比深色要低一些。饱和度还和亮度有关，因为若在饱和的彩色光中增加白光的成分，由于增加了光能，因而变得更亮了，但是它的饱和度却降低了。如果在某色调的彩色光中掺入别的彩色光，会引起色调的变化，掺入白光时仅引起饱和度的变化。

2．三原色原理

三原色原理是色度学中最基本的原理，是指自然界常见的各种颜色光，都可由红（R）、绿（G）、蓝（B）3 种颜色按不同比例相配制而成；同样绝大多数颜色光也可以分解成红、绿、蓝三种色光。当然三原色的选择并不是唯一的，也可以选择其他 3 种颜色为三原色，但是，3 种颜色必须是相互独立的，即任何一种颜色都不能由其他两种颜色合成。由于人眼对红、绿、蓝 3 种色光最敏感，因此，由这 3 种颜色相配制所得的彩色范围也最广，所以一般都选用这 3 种颜色作为基色。

3．彩色空间

- RGB彩色空间：在多媒体计算机技术中，用得最多的是RGB彩色空间表示。因为计算机的彩色监视器的输入需要R、G、B 3个彩色分量，通过3个分量的不同比例，在显示屏幕上可以合成所需要的任意颜色，所以不管多媒体系统采用什么形式的彩色空间表示，最后的输出一定要转换成RGB彩色空间表示。
- YUV彩色空间：在现代彩色电视系统中，通常采用三管彩色摄像机或彩色CCD摄像机，把摄得的彩色图像信号经分色棱镜分成R0、G0、B0 3个分量的信号；分别经放大和校正得到三基色，再经过矩阵变换电路得到亮度信号Y、色差信号R－Y和B－Y，最后发送端将Y、R－Y和B－Y 3个信号进行编码，用同一信道发送出去，这就是我们常用的YUV彩色空间。
- CMYK彩色空间：CMYK也称作印刷色彩模式，是一种依靠反光的色彩模式，与RGB类似，C、M、Y是3种印刷油墨名称的首字母，青色Cyan、品红色Magenta、黄色Yellow。其中K是源自一种只使用黑墨的印刷版Key Plate。从理论上来说，只需要C、M、Y三种油墨就足够了，它们三个加在一起就应该得到黑色。但是由于目前制造工艺还不能造出高纯度的油墨，C、M、Y相加的结果实际是一种暗红色。

6.2 数据压缩标准

1．H.261

H.261 是国际电联 ITU-T 的一个标准草案，H.261 又称为 P*64，其中 P 为 64Bb/s 的取值范围，是 1～30 的可变参数，它最初是针对在 ISDN 上实现电信会议应用特别是面对面的可视电话和视频会议而设计的。实际的编码算法类似于 MPEG 算法，但不能与后者兼容。

H.261在实时编码时比MPEG所占用的CPU运算量少得多，此算法为了优化带宽占用量，引进了在图像质量与运动幅度之间的平衡折中机制，也就是说，剧烈运动的图像比相对静止的图像质量要差。因此这种方法是属于恒定码流可变质量编码而非恒定质量可变码流编码。

2．H.263

H.263 是国际电联 ITU-T 的一个标准草案，是为低码流通信而设计的。但实际上这个标准可用于很宽的码流范围，而非只用于低码流应用，它在许多应用中可以被认为用于取代 H.261。H.263 的编码算法与 H.261 一样，但做了一些改善和改变，以提高性能和纠错能力。H.263 标准在低码率下能够提供比 H.261 更好的图像效果。H.263 支持 5 种分辨率，即除支持 H.261 中所支持的 QCIF 和 CIF 外，还支持 SQCIF、4CIF 和 16CIF，SQCIF 相当于 QCIF 一半的分辨率，而 4CIF 和 16CIF 分别为 CIF 的 4 倍和 16 倍。

1998 年 IUT-T 推出的 H.263+是 H.263 建议的第二版，它提供了 12 个新的可协商模式和其他特征，允许使用更多的源格式，图像时钟频率也有多种选择，拓宽应用范围；另一重要的改进是可扩展性，它允许多显示率、多速率及多分辨率，增强了视频信息在易误码、易丢包、异构网络环境下的传输。H.263 已经基本上取代了 H.261。

3．M-JPEG

M-JPEG（Motion-Join Photographic Experts Group）技术即运动静止图像（或逐帧）压缩技术，广泛应用于非线性编辑领域，可精确到帧编辑和多层图像处理，把运动的视频序列作为连续的静止图像来处理，这种压缩方式单独完整地压缩每一帧，在编辑过程中可随机存储每一帧，可进行精确到帧的编辑，此外 M-JPEG 的压缩和解压缩是对称的，可由相同的硬件和软件实现。但 M-JPEG 只对帧内的空间冗余进行压缩。不对帧间的时间冗余进行压缩，故压缩效率不高。

M-JPEG 标准所根据的算法是基于 DCT（离散余弦变换）和可变长编码。M-JPEG 的关键技术有变换编码、量化、差分编码、运动补偿、霍夫曼编码和游程编码等。M-JPEG 的优点是可以很容易做到精确到帧的编辑，设备比较成熟；缺点是压缩效率不高。

4．MPEG-1

MPEG 是活动图像专家组（Moving Picture Experts Group）的缩写，MPEG 组织最初得到的授权是制订用于“活动图像”编码的各种标准，随后扩充为“伴随的音频”及其组合编码。后来针对不同的应用需求，解除了“用于数字存储媒体”的限制，成为现在制订“活动图像和音频编码”标准的组织。

MPEG-1 标准于 1993 年 8 月公布，用于传输 1.5Mb/s 数据传输率的数字存储媒体运动图像及其伴音的编码。该标准包括 5 个部分：第一部分说明了如何根据第二部分（视频）及第三部分（音频）的规定，对音频和视频进行复合编码。第四部分说明了检验解码器或编码器的输出比特流符合前三部分规定的过程。第五部分是一个用完整的 C 语言实现的编码和解码器。

5．MPEG-2

MPEG 组织于 1994 年推出 MPEG-2 压缩标准，以实现视/音频服务与应用互操作的可能性。MPEG-2 标准是针对标准数字电视和高清晰度电视在各种应用下的压缩方案和系统层的详细规定，编码率为 3Mb/s～100Mb/s，标准的正式规范在 ISO/IEC13818 中。MPEG-2

不是 MPEG-1 的简单升级，在系统和传送方面做了更加详细的规定和进一步的完善，特别适用于广播级的数字电视的编码和传送，被认定为 SDTV 和 HDTV 的编码标准。

MPEG-2 图像压缩的原理是利用图像中的两种特性：空间相关性和时间相关性。这两种相关性使得图像中存在大量的冗余信息。如果我们能将这些冗余信息去除，只保留少量非相关信息进行传输，就可以大大节省传输频带。而接收机利用这些非相关信息，按照一定的解码算法，可以在保证一定的图像质量的前提下恢复原始图像。一个好的压缩编码方案就是能够最大限度地去除图像中的冗余信息。

MPEG-2 的编码图像可分为 3 类，分别称为 I 帧、P 帧和 B 帧。I 帧图像采用帧内编码方式，即只利用单帧图像内的空间相关性，而没有利用时间相关性。P 帧和 B 帧图像采用帧间编码方式，即同时利用空间和时间上的相关性。P 帧图像只采用前向时间预测，可以提高压缩效率和图像质量。P 帧图像中可以包含帧内编码的部分，即 P 帧中的每一个宏块可以是前向预测，也可以是帧内编码。B 帧图像采用双向时间预测，可以大大提高压缩倍数。

为更好地表示编码数据，MPEG-2 用句法规定了一个层次性结构。它分为 6 层，自上到下分别是：图像序列层、图像组（GOP）、图像、宏块条、宏块、块。

6．MPEG-4

MPEG 组织于 1999 年 2 月正式公布了 MPEG-4（ISO/IEC14496）标准第一版本。同年年底发布 MPEG-4 第二版，且于 2000 年年初正式成为国际标准。

MPEG-4 与 MPEG-1 和 MPEG-2 有很大的不同。MPEG-4 不只是具体压缩算法，它是针对数字电视、交互式绘图应用（影音合成内容）、交互式多媒体（WWW、资料撷取与分散）等整合及压缩技术的需求而制定的国际标准。MPEG-4 标准将众多的多媒体应用集成于一个完整的框架内，旨在为多媒体通信及应用环境提供标准的算法及工具，从而建立起一种能被多媒体传输、存储、检索等应用领域普遍采用的统一数据格式。

MPEG-4 标准同以前标准的最显著的差别在于它是采用基于对象的编码理念，即在编码时将一幅景物分成若干在时间和空间上相互联系的视频、音频对象，分别编码后，再经过复用传输到接收端，然后再对不同的对象分别解码，从而组合成所需要的视频和音频。MPEG-4 系统的一般框架是：对自然或合成的视听内容的表示；对视听内容数据流的管理，如多点、同步、缓冲管理等；对灵活性的支持和对系统不同部分的配置。

与 MPEG-1、MPEG-2 相比，MPEG-4 具有如下独特的优点：基于内容的交互性；高效的压缩性；通用的访问性。MPEG-4 提供了易出错环境的鲁棒性，来保证其在许多无线和有线网络及存储介质中的应用，此外，MPEG-4 还支持基于内容的可分级性，即把内容、质量、复杂性分成许多小块来满足不同用户的不同需求，支持具有不同带宽、不同存储容量的传输信道和接收端。

MPEG-4 的主要应用领域有：因特网多媒体应用；广播电视；交互式视频游戏；实时可视通信；交互式存储媒体应用；演播室技术及电视后期制作；采用面部动画技术的虚拟会议；多媒体邮件；移动通信条件下的多媒体应用；远程视频监控；通过 ATM 网络等进行的远程数据库业务等。

7．MPEG-7

MPEG-7 标准被称为“多媒体内容描述接口”，为各类多媒体信息提供一种标准化的描

述，这种描述将与内容本身有关，允许快速和有效地查询用户感兴趣的资料。它将扩展现有内容识别专用解决方案的有限能力，特别是它还包括了更多的数据类型。换言之，MPEG-7规定一个用于描述各种不同类型多媒体信息的描述符的标准集合，该标准于 1998 年 10 月提出。

MPEG-7 的目标是支持多种音频和视觉的描述，包括自由文本、*N* 维时空结构、统计信息、客观属性、主观属性、生产属性和组合信息。对于视觉信息，描述将包括颜色、视觉对象、纹理、草图、形状、体积、空间关系、运动及变形等。

MPEG-7 的目标是根据信息的抽象层次，提供一种描述多媒体材料的方法，以便表示不同层次上的用户对信息的需求。以视觉内容为例，较低抽象层将包括形状、尺寸、纹理、颜色、运动（轨道）和位置的描述。对于音频的较低抽象层包括音调、调试、音速、音速变化、音响空间位置。最高层将给出语义信息：如“这是一个场景：一只鸭子正躲藏在树后并有一辆汽车正在幕后通过”。抽象层与提取特征的方式有关：许多低层特征能以完全自动的方式提取，而高层特征需要更多人的交互作用。MPEG-7 还允许依据视觉描述的查询去检索声音数据，反之也一样。

MPEG-7 的目标是支持数据管理的灵活性、数据资源的全球化和互操作性。

MPEG-7 标准化的范围包括：一系列的描述子（描述子是特征的表示法，一个描述子就是定义特征的语法和语义学）；一系列的描述结构（详细说明成员之间的结构和语义）；一种详细说明描述结构的语言、描述定义语言（DDL）；一种或多种编码描述方法。

MPEG-7 标准可以支持非常广泛的应用，具体如下：音视数据库的存储和检索；广播媒体的选择（广播、电视节目）；因特网上的个性化新闻服务；智能多媒体、多媒体编辑；教育领域的应用（如数字多媒体图书馆等）；远程购物；社会和文化服务（历史博物馆、艺术走廊等）；调查服务（人的特征的识别、辩论等）；遥感；监视（交通控制、地面交通等）；生物医学应用；建筑、不动产及内部设计；多媒体目录服务（如黄页、旅游信息、地理信息系统等）；家庭娱乐（个人的多媒体收集管理系统等）。

8．MPEG-21

制定 MPEG-21 标准的目的是：①将不同的协议、标准、技术等有机地融合在一起；②制定新的标准；③将这些不同的标准集成在一起。MPEG-21 标准其实就是一些关键技术的集成，通过这种集成环境对全球数字媒体资源进行透明和增强管理，实现内容描述、创建、发布、使用、识别、收费管理、产权保护、用户隐私权保护、终端和网络资源抽取、事件报告等功能。

任何与 MPEG-21 多媒体框架标准环境交互或使用 MPEG-21 数字项实体的个人或团体都可以看作用户。从纯技术角度来看，MPEG-21 对于“内容供应商”和“消费者”没有任何区别。MPEG-21 多媒体框架标准包括如下用户需求：内容传送和价值交换的安全性；数字项的理解；内容的个性化；价值链中的商业规则；兼容实体的操作；其他多媒体框架的引入；对 MPEG 之外标准的兼容和支持；一般规则的遵从；MPEG-21 标准功能及各个部分通信性能的测试；价值链中媒体数据的增强使用；用户隐私的保护；数据项完整性的保证；内容与交易的跟踪；商业处理过程视图的提供；通用商业内容处理库标准的提供；长线投资时商业与技术独立发展的考虑；用户权利的保护，包括：服务的可靠性、债务与保险、损失与破坏、付费处理与风险防范等；新商业模型的建立和使用。

9．DVI

DVI 视频图像压缩法是 Intel 公司推出的一个压缩算法，其性能与 MPEG-1 相当，即图像质量可达到 VHS 的水平。压缩后的图像数据率约为 1.5Mb/s。应用 Intel 公司生产的 i750 芯片组，即 82750PB 和 82750DB 可实时完成 DVI 视频图像的编码和解码算法。

为了扩大 DVI 技术的应用，Intel 公司又推出了 DVI 算法软件解码算法，称为 Indeo 技术。它能将未压缩的数字视频文件压缩为 1/5～1/10。Indeo 技术已被附加在某些产品中，如微软公司 Video for Windows 和苹果公司的 Quicktime。

Indeo 技术使用多类有损和无损压缩技术。Indeo 技术在视频捕获卡记录的同时实时地对它进行压缩，因此未压缩的数据无须存在盘上。从视频摄像机、VCR 或激光盘上接收到的任何标准格式（如 NTSC 存在的视频）都由视频捕获卡（如 Intel 的 Smart Video Recorder Board）转换为数字格式。

6.3　图形图像

在计算机科学中，图形和图像这两个概念是有区别的：图形一般指用计算机绘制的画面，如直线、圆、圆弧、任意曲线和图表等；图像则是指由输入设备捕捉的实际场景画面或以数字化形式存储的任意画面。

图像是由一些排成行列的像素组成的，在计算机中的存储格式有 BMP、PCX、TIF、GIFD 等，一般数据量都较大。它除可以表达真实的照片外，也可以表现复杂绘画的某些细节，并具有灵活和富有创造力等特点。

与图像文件不同，在图形文件中只记录生成图的算法和图上的某些特征点，也称为矢量图。在计算机还原输出时，相邻的特征点之间用特定的很多段小直线连接就形成曲线，若曲线是一条封闭的图形，也可靠着色算法来填充颜色。它的最大优点是容易进行移动、缩放、旋转和扭曲等变换，主要用于表示线框型的图画、工程制图、美术字等。常用的矢量图形文件有 3DS（用于 3D 造型）、DXF（用于 CAD）、WMF（用于桌面出版）等。图形只保存算法和特征点，所以相对于位图的大数据量来说，它占用的存储空间也较小。但由于每次屏幕显示时都需重新计算，故显示速度没有图像快。另外，在打印输出和放大时，图形的质量较高而点阵图常会发生失真。

下面为了叙述的方便，不再区分图形和图像。

图形的主要指标为分辨率、色彩数与灰度。分辨率一般有屏幕分辨率和输出分辨率两种，前者用每英寸行数与列数表示，数值越大，图形质量越好；后者衡量输出设备的精度，以每英寸的像素点数表示，数值越大越好。如果一个图形是 16 位图像，则颜色数为 2 的 16 次方，共可表现 65 536 种颜色。当图形达到 24 位时，可表现 1 677 万种颜色，即真彩。常见的色彩位表示一般有 2 位、4 位、8 位、16 位、24 位、32 位、64 位等。

常见的图形有如下几种。

- BMP：PC机上最常见的位图格式，有压缩和不压缩两种形式。BMP格式可表现从2位到24位的色彩，分辨率为从480像素×320像素至1024像素×768像素。该格式在Windows环境下相当稳定，所以在对文件大小没有限制的场合中运用最为广泛。
- DIB：描述图像的能力基本与BMP相同，并且能运行于多种硬件平台，只是文件较大。

- PCX：是由Zsoft 公司创建的一种经过压缩且节约磁盘空间的PC位图格式，它最高可表现24位图形。过去有一定的市场，但随着JPEG的兴起，其地位已逐渐降低。
- DIF：AutoCAD中的图形文件，它以ASCII方式存储图形，表现图形在尺寸大小方面十分精确，可以被CorelDraw、3DS等软件调用编辑。
- WMF：Microsoft Windows图元文件，具有文件短小、图案造型化的特点，整个图形内容常由各独立组成部分拼接而成。但该类图形比较粗糙，并只能在Microsoft Office中调用编辑。
- GIF：GIF（Graphics Interchange Format）的原义是“图像互换格式”，是CompuServe公司在 1987年开发的图像文件格式。GIF文件的数据，是一种基于LZW算法的连续色调的无损压缩格式。其压缩率一般在50%左右，它不属于任何应用程序。目前几乎所有相关软件都支持它，公共领域有大量的软件在使用GIF图像文件。GIF图像文件的数据是经过压缩的，而且采用了可变长度等压缩算法。GIF格式的另一个特点是其在一个GIF文件中可以存多幅彩色图像，如果把存于一个文件中的多幅图像数据逐幅读出并显示到屏幕上，就可构成一种最简单的动画。
- JPEG：JPEG格式可以大幅度地压缩图形文件。同样一幅画面，用JPEG格式存储的文件是其他类型图形文件的1/10～1/20，而色彩数最高可达到24位，所以它被广泛运用于Internet上，以节约网络传输资源。JPEG文件之所以较小，是以损失图像质量为代价的。
- PSD：Photoshop 中的标准文件格式，专门为Photoshop而优化。
- CDR：CorelDraw的文件格式。
- PCD：Photo CD格式，由Kodak公司开发，其他软件系统对其只能读取。
- TIFF：标签图像文件格式，它是一种主要用来存储包括照片和艺术图在内的图像的文件格式。它最初由 Aldus公司与微软公司一起为PostScript打印开发。

6.4 音频

用计算机处理声音归结为语音合成、存储和输出等技术。

语音合成技术可分为发音参数合成、声道模型参数合成和波形编辑合成，语音合成策略可分为频谱逼近和波形逼近。

发音参数合成对人的发音过程进行直接模拟，定义了唇、舌、声带的相关参数，由这些发音参数估计声道截面积函数，进而计算声波。但由于人发音生理过程的复杂性，理论计算与物理模拟之间存在差异，合成语音的质量暂时还不理想。声道模型参数语音合成方法基于声道截面积函数或声道谐振特性合成语音，这类合成器的比特率低，音质适中。波形编辑语音合成技术基于时域波形修改的语音合成技术，直接把语音波表数据库中的波形级联起来，输出连续语流。这种语音合成技术用原始语音波形替代参数，而且这些语音波形取自自然语音的词或句子，它隐含了声调、重音、发音速度的影响，合成的语音清晰自然。其质量普遍高于参数合成。

推动喇叭发声的电信号是连续的模拟信号。计算机只能存储数字信号，模拟信号转换成数字信号包括采样和量化两个过程。采样是在一系列离散的时间点上测量模拟信号的大小，而量化则是用数字量来表示该大小。

实现计算机语音输出有两种方法：一是录音/重放，二是文－语转换。若采用第一种方法，首先要把模拟语音信号转换成数字序列，编码后暂存于存储设备中（录音），需要时再经解码，重建声音信号（重放）。录音/重放可获得高音质声音，并能保持特定人或乐器的音色。但所需的存储容量随发音时间线性增长。

第二种方法是基于声音合成技术的一种声音产生技术，它可用于语音合成和音乐合成。文－语转换是语音合成技术的延伸，它能把计算机内的文本转换成连续自然的语声流。若采用这种方法输出语音，应预先建立语音参数数据库、发音规则库等。需要输出语音时，系统按需求先合成出语音基元，再按语音学规则或语言学规则，连接成自然的语声流。文－语转换的参数库不随发音时间增长而加大，而规则库却随语音质量的要求而增大。

常见的音频格式如下。

- WAVE：WAVE格式的声音文件的扩展名为WAV，这种格式记录了声音的波形，即模拟信号的采样数值。WAV文件所记录的声音文件能够和原声基本一致。在播放WAV文件时，只需进行数字模拟转换，将数字量转换成相应的电信号值并构成模拟信号即可推动喇叭发音。从理论上说，采样率达44kHz（每秒采样44 000次）、采样字节长度达16位的音质已能和常规CD唱片相当。因为WAVE格式要把声音的每个细节都记录下来，而且不压缩，所以它的文件很大。例如，如果采样率为44kHz，那么每一秒钟就有44K×16×2（立体声）=1 441 792位产生，那么，一张650MB的空白光盘最多也只能容纳五六十分钟的节目。
- MOD：MOD格式的声音文件的扩展名可为MOD、ST3、XT、S3M和FAR的任意一种。MOD及播放器大约起源于20世纪80年代初，原先是作为软声卡问世的，MOD只是这类音乐文件的总称。MOD格式的文件中不仅存放了乐谱（最初只能支持4个声道，到现在已有16，甚至32个声道的文件及播放器），而且存放了乐曲使用的各种音色样本。由于制作人创作歌曲使用的音色样本同听众回放文件时使用的音乐样本完全相同，所以这样的文件有几个显著的优点：回放效果明确；音色种类永无止境。
- MP3：MP3记录了音乐经数字比压缩的编码，压缩较大，在网络、可视电话通信方面，大有用武之地。但MP3的失真较大。在播放MP3文件时，需要相应的解码器将它转换成模拟信号的数字序列，再经数字模拟转换推动喇叭发音。
- Real Audio：Real Audio格式的声音文件的扩展名为RA，Real Audio也是为了解决网络传输带宽资源而设计的，因此主要目标是压缩比和容错性，其次才是音质。Real Audio压缩比很大，相对而言，Real Audio的音质量比MPEG-3好。
- CD Audio：CD Audio格式的声音文件的扩展名为CDA，回放和采样字节都是16位，现在有些厂家在录制CD时采用20位录音，这样就产生了一些耳朵听不到但大脑感觉得到的波形，可谓CD中的精品。CDA的缺点是：无法编辑，文件太大。
- MIDI：MIDI格式的声音文件的扩展名是MID。乐器数字接口（Musical Instrument Digital Interface，MIDI）泛指数字音乐的国际标准，它始创于1982年。MIDI描述了音乐演奏过程的指令，利用MIDI文件演奏音乐，所需的存储量最少。MIDI标准规定了不同厂家的电子乐器与计算机连接的电缆和硬件。作为音乐工业的数据通信标准，MIDI是一种非常专业的语言，它能指挥各音乐设备的运转，而且具有统一的标准格式，能够模仿原始乐器的各种演奏技巧甚至无法演奏的效果。MIDI依赖于回放设备，为了避免这种缺点，网络上出现了“软波表”之类的软音源。采用专业音源

的波表，利用CPU对网络上传来的短短的MIDI数据进行回收，其效果能够被制作者预测。MIDI的另一个缺点就是不能记录人声等声音。

6.5 视频

动态图像，包括动画和视频信息，是连续渐变的静态图像或图形序列沿时间轴顺次更换显示，从而构成运动视感的媒体。当序列中每帧图像是由人工或计算机产生的图像时，常称为动画；当序列中每帧图像是通过实时摄取自然景象或活动对象时，常称为影像视频，或简称视频。

视频信息在计算机中存放具体格式有很多，常见的有如下几种。

- Quicktime：苹果公司的产品，采用了面向最终用户桌面系统的低成本、全运动视频的方式，在软件压缩和解压缩中也开始采用这种方式。向量量化是Quicktime的软件压缩技术之一，它在最高为30帧/s下提供的视频分辨率是320像素×240像素，而且不用硬件帮助。向量量化预计可成为全运动视频的主要技术，向量量化方法达到的压缩比例为25:1～200:1。其视频信息采用MOV或QT文件格式。
- AVI：微软公司的视频格式。音频视频交错（AVI）也是桌面系统上低成本低分辨率的视频格式，AVI可在160×120的视窗中以15帧/s回收视频并可带有8位的声音，也可以在VGA或超级VGA监视器上回收。与超过320线的VCR分辨率相比，这一分辨率明显低于正常电视信号的分辨率。AVI很重要的一个特点是可伸缩性，使用AVI算法的性能依赖于它一起使用的基础硬件。AVI包括了几种基于软件的压缩和解压缩算法，其中某些算法被优化用于运动视频，其他算法则被优化用于静止视频。
- RealMedia：RealNetworks公司所制定的音频/视频压缩规范，采用了流的方式播放，使用户可以边下载边播放，而且其极高的影像压缩率虽然牺牲了一些画质与音质，但却能在较慢的网速上流畅地播放RealMedia格式的音乐和视频。RealMedia是目前Internet上最流行的跨平台的客户/服务器结构多媒体应用标准，其采用音频/视频流和同步回放技术实现了网上全带宽的多媒体回放。在RealMedia规范中主要包括3类文件：RealAudio（用以传输接近CD音质的音频数据）、RealVideo（用来传输连续视频数据）和RealFlash（RealNetworks公司与Macromedia公司合作推出的新一代高压缩比动画格式）。其文件格式通常为RA或RM，一张用RM格式压缩的光盘上可以存放4部电影。RealPlayer是RealMedia的播放工具，利用Internet资源对这些符合RealMedia技术规范的音频/视频进行实况转播。
- ASF：Advanced Streaming Format（高级流格式）的缩写，是微软公司为了和RealMedia竞争而发展出来的一种可以直接在网上观看视频节目的文件压缩格式。由于它使用了MPEG-4的压缩算法，所以压缩率和图像的质量都很不错。因为ASF是以一个可以在网上即时观赏的视频流格式存在的，所以它的图像质量比VCD差，但比同是视频流格式的RealMedia格式要好。
- WMV：一种独立于编码方式的、在Internet上实时传播多媒体的技术标准，微软公司希望用其取代QuickTime之类的技术标准，以及WAV、AVI之类的文件扩展名。WMV的主要优点包括：本地或网络回放、可扩充的媒体类型、部件下载、可伸缩的媒体类型、流的优先级化、多语言支持、环境独立性、丰富的流间关系，以及扩展性等。

第 7 章

计算机的体系结构和主要部件

也许有人认为软件设计师似乎不需要什么硬件的知识，计算机的硬件理论知识并非是空中楼阁，不是好看或者用来考试的，它是确确实实的每台计算机设计和制造的基础，而且在学习的过程中，我们能够发现许多在硬件上使用的原则在软件上使用同样有益。在很多时候，计算机软件和硬件并非截然分开，而是有一个此消彼长、相互促进的发展过程。比如早期的中央处理器没有浮点运算功能，浮点运算需要使用软件实现，而后来，许多处理器都内置了浮点运算功能。另外，嵌入式系统的软件设计师必须考虑硬件的问题。

7.1 机内代码及运算

众所周知，计算机只处理二进制数据，二进制是最简单的进制方式，只有 0 和 1 两个基数，也就是说，计算机底层硬件只要能保持两个状态即可，这样使得计算机的底层设计变得简单，出错的概率也大为减小。当然二进制数据使得表示和保存数据的长度大大增加，但是大规模和超大规模的集成电路使得这成为次要的问题，人们能在越来越小的芯片空间里容纳越来越多的电路。

另一方面计算机为了使得处理方便，其内部存储数据的格式和我们看见的有所不同。

7.1.1 数的进制

1．进制的表示法

R 进制，通常说法就是逢 R 进 1。可以用的数为 R 个，分别是 0，1，2，…，$R-1$。例如十进制数的基数为 10，即可以用到的数码个数为 10，它们是 0，1，2，3，4，5，6，7，8，9。二进制数的基数为 2，可用的数码个数为 2，它们是 0 和 1。

为了把不同的进制数分开表示，避免造成混淆，采用下标的方式来表示一个数的进制，如十进制数 56 表示为：$(56)_{10}$，八进制数 42 表示为：$(42)_8$。

对于任意一个 R 进制数，它的每一位数值等于该位的数码乘以该位的权数。权数由一个幂 R^k 表示，即幂的底数是 R，指数为 k，k 与该位和小数点之间的距离有关。当该位位于小数点左边，k 值是该位和小数点之间数码的个数，而当该位位于小数点右边，k 值是负值，其绝对值是该位和小数点之间数码的个数加 1。

例如，十进制数 1234.56，其数值可计算如下：

$1234.56 = 1\times10^3+2\times10^2+3\times10^1+4\times10^0+5\times10^{-1}+6\times10^{-2}$

例如，二进制数 l0100.01 的值可计算如下：

$l0100.01 = 1\times 2^4+1\times 2^2+1\times 2^{-2}$

2．进制的转换

1）*R* 进制数转换成十进制数

按照上述表示法，即可计算出 *R* 进制数十进制的值。

2）十进制数转换为 *R* 进制数

最常用的是“除以 *R* 取余法”。例如，将十进制数 94 转换为二进制数：

```
2 | 94        余  0
2 | 47            1
2 | 23            1
2 | 11            1
2 | 5             1
2 | 2             0
    1             1
```

将所得的余数从低位到高位排列$(1011110)_2$就是 94 的二进制数。

3）二进制数与八进制数、十六进制数之间的转换

将二进制数转换为八进制数，只有将每 3 个二进制数转换为八进制数即可，将二进制数转换为十六进制数，只要将每 4 个二进制数转换为八进制数即可。将八进制数转换为二进制数，只要将每个八进制数转换为 3 位二进制数即可，将十六进制数转换为二进制数，只要将每个十六进制数转换为 4 位二进制数即可。上面的转换都是以小数点作为计算数码个数的起点。八进制数和十六进制数转换可先转换为二进制数，然后再转换为目标进制。

7.1.2 原码、反码、补码、移码

一个正数的原码、反码、补码是相同的，负数则不同。先提一个问题，为什么在计算机中要使用这些编码方式呢？

1．原码

将最高位用作符号位（0 表示正数，1 表示负数），其余各位代表数值本身的绝对值的表示形式。这种方式是最容易理解的。

例如，+11 的原码是 00001011，－11 的原码是 10001011。

但是直接使用原码在计算时却会有麻烦，比如$(1)_{10}+(-1)_{10}=0$，如果直接使用原码则：

$(00000001)_2+(1000001)_2=(10000010)_2$

这样计算的结果是－2，也就是说，使用原码直接参与计算可能会出现错误的结果。所以，原码的符号位不能直接参与计算，必须和其他位分开，这样会增加硬件的开销和复杂性。

2．反码

正数的反码与原码相同。负数的反码符号位为 1，其余各位为该数绝对值的原码按位取反。这个取反的过程使得这种编码称为“反码”。

例如，－11 的反码：11110100

同样对上述加法，使用反码的结果是：

$(00000001)_2 + (11111110)_2 = (11111111)_2$

这样的结果是负 0，而在人们普遍的观念中，0 是不分正负的。反码的符号位可以直接参与计算，而且减法也可以转换为加法计算。

3．补码

正数的补码与原码相同。负数的补码是该数的反码加 1，这个加 1 就是“补”。

例如，－11 的补码：11110100+1 = 11110101

再次做加法是这样的：

$(00000001)_2 + (11111111)_2 = (00000000)_2$

直接使用补码进行计算的结果是正确的。注意，这里只是举例，并非证明。

对一个补码表示的数，要计算其原码，只要对它再次求补，可得该数的原码。

由于补码能使符号位与有效值部分一起参加运算，从而简化运算规则，同时它也使减法运算转换为加法运算，进一步简化计算机中运算器的电路，这使得在大部分计算机系统中，数据都使用补码表示。

4．移码

无论正数还是负数，移码都是在补码的基础上，对符号位取反。

例如，

11 的补码是 00001011，所以它的移码为：10001011。

－11 的补码是 11110101，所以它的移码为：01110101。

再次做加法是这样的：

$(10001011)_2 + (01110101)_2 = (00000000)_2$

直接使用移码进行计算的结果也是正确的。

7.1.3　定点数和浮点数

定点数和浮动数的区别在于如何对待小数点，在运算方式上也不相同，衡量一个计算机系统，定点运算和浮点运算是两个重要的指标。定点数的小数点是隐含的，固定在某个位置。如果该位置是在数的最低位之后，就是定点整数。定点数表示比较简单，运算规则也比较容易实现，但是当数值范围变化大时，使用定点数表示和运算就比较困难。

为了表示更大范围的数值，可以使用浮点数表示法。

在表示一个很大的数时，常常使用一种称为科学计数法的方式：

$N = M{*}R^{e}$

其中 M 称为尾数，e 是指数，R 为基数。

浮点数就是使用这种方法来表示大范围的数，其中指数一般是2、8、16。而且对于特定机器而言，指数是固定不变的，所以在浮点数中指数并不出现。从这个表达式可以看出：浮点数表示的精读取决于尾数的宽度，范围取决于基数的大小和指数的宽度。

1．格式化数

使用格式化数是提高浮点数有效位的方法。格式化的意思是把尾数前面加 0，同时修改指数，这样在尾数位数固定的情况下，能提供最多的有效位来表示尾数。当指数小于能够表示的最小值时，这个数称为机器零，此时会把尾数和指数同时清零。看到这里，读者应该能回答指数为什么常使用移码来表示问题。

2．定点数的算术运算和溢出处理

如前所述，计算机中通常使用补码进行计算。两个正数相加，如果结果的符号位变成1，则表示有溢出；同样两个负数相加，如果结果的符号位变成了 0，那么也意味着溢出。如果是正数和负数相加，则不会出现溢出的情况。

判断处理的方法可以再增加一个符号位，称为第一符号位，原来那个符号位变成第二符号位。计算时两个符号位都参与计算，如果计算结果的两个符号位相同，表示没有溢出；如果不同，就表示出现了溢出。而第一符号位才是真正的符号。

也可以通过进位信号来判断，当结果的最高位和符号位的进位信号一致时（都有进位信号或都没有进位信号），则没有溢出，否则表示有溢出。

3．定点数的逻辑运算

逻辑运算意味着各位的运算不产生进位，操作数的对应位独立计算。逻辑加实际就是按位“或”的计算，逻辑乘实际上是按位“与”的操作，逻辑非是按位“取反”。在校验码中，我们将接触到逻辑运算。

4．浮点数的运算

浮动数运算过程比定点数复杂，包括如下过程。

1）对阶

首先计算两个数的指数差，把指数小的向指数大的对齐，并将尾数右移指数差的位数，这样两个浮点数就完成了对阶的操作。可以看出，对阶的过程可能使得指数小的浮点数失去一些有效位。如果两个浮点数阶数相差很大，大于指数小的浮点数的尾数宽度，那么对阶后那个浮点数的尾数就变成了 0，即当作机器零处理。

2）尾数计算

对阶完成后，两个浮点数尾数就如同定点数，计算过程同定点数计算。

3）结果格式化

尾数计算后，可能会产生溢出，此时将尾数右移，同时指数加 1，如果指数加 1 后发生了溢出，则表示两个浮点数的运算发生了溢出。

如果尾数计算没有溢出，则尾数不断左移，同时指数减 1，直到尾数为格式化数。如果这个过程中，指数小于机器能表达的最小数，则将结果置“机器零”，这种情况称为下溢。

7.1.4 校验码概述

1．编码体系

体系这个词总是比较高深，但在这里有大材小用之嫌，编码体系指一种编码方式中所有合法码字的集合。合法码字占所有码字的比率就是编码效率。读者可计算一下 BCD 编码的编码效率。

2．码距

码距是衡量一种编码方式的抗错误能力的一个指标。数字信息在传输和存取的过程中，由于各种意外情况的发生，数据可能会发生错误，即所谓误码。

一种编码，如果所有可能的码字都是合法码字，如 ASCII，当码字中的一位发生错误时，这个错误的码仍然在编码体系中，这样我们称这种编码的码距小。如果把编码体系变得稀疏一点，使得很多的信号值不在编码体系之内，这样，合法的码字如果出现错误，可能就变成了不合法的编码，这样编码的码距就大。

定义：一个编码系统中任意两个合法的编码之间的不同的二进制位称为这两个码字的码距。该编码系统的任意两个编码之间的距离的最小值称为该编码系统的码距。

显然，码距越大，编码系统的抗偶然错误能力越强，甚至可以纠错（纠错详见各种编码的介绍）。同时，码距的增加，使得必须提供更多的空间来存放码字，数据冗余增加，编码效率则降低了，系统设计师需要综合考虑系统效率和系统健壮性两个方面，在众多的编码体系中选择适合特定目标系统的编码。

7.1.5 奇偶校验

奇偶校验较为简单，被广泛地采用，常见的串口通信中基本都使用奇偶校验作为数据校验的方法。

一个码距为 1 的编码系统加上一位奇偶校验码后，码距就成为 2。产生奇偶校验时将信息数据的各位进行模二加法，直接使用这个加法的结果的称为奇校验。把这个加法值取反后作为校验码的称为偶校验。从直观的角度而言，奇校验的规则是：信息数据中各位中 1 的个数为奇数，校验码为 1，否则校验码为 0，偶校验则相反。

使用一位奇偶校验的方法能够检测出一位错误，但无法判断是哪一位出错。当两位同时出错时，奇偶校验也无法检测出来。所以奇偶校验通常用于对少量数据的校验，如一个字节。在串口通信中，通常是一个字节带上起始位、结束位和校验位共 11 位来传送。

如果对一位奇偶校验进行扩充，在若干个带有奇偶校验码的数据之后，再附上一个纵向的奇偶校验数据，如表 7-1 所示。

表 7-1　奇偶校验位组成

信　息　位				校　验　位
α_1	α_{-2}	…	α_{-m}	Hp_1
β_1	β_2		β_m	Hp_2
...				
n_1	n_2		n_3	Hp_n
Vp_1	Vp_2		Vp_m	Hp_{m+1}

这样，在出现一个错误的情况下，就能找到这个错误。而如果出现两个以上的错误，则可能无法判断误码的位置。这种方式在移动通信领域中被广泛采用。

7.1.6 海明码

海明码是奇偶校验的另一种扩充。和上面提到的奇偶校验不同之处在于海明码采用多位校验码的方式，在这些校验位中的每一位都对不同的信息数据位进行奇偶校验，通过合理地安排每个校验位对原始数据进行校验位组合，可以达到发现错误、纠正错误的目的。

假设数据位有 m 位，如何设定校验位 k 的长度才能满足纠正一位错误的要求呢？下面做一个简单的推导。

k 位的校验码可以有 2^k 个值。显然，其中一个值表示数据是正确的，而剩下的 2^k-1 个值意味着数据中存在错误，如果能够满足：$2^k-1>m+k$ （$m+k$ 为编码后的总长度），在理论上 k 个校验码就可以判断是哪一位（包括信息码和校验码）出现问题。

校验方程是指示每个校验位对哪些信息位进行校验的等式。

确定了 k 的值后，如何确定每 k 位中的每一位对哪些数据进行校验呢？上面的推导只是说能够做，那么如何达到纠错的目的呢？但是幸好考试中都会列出海明校验方程。例如：

$b_1\oplus b_3\oplus b_5\oplus b_7=0$ ①

$b_2\oplus b_3\oplus b_6\oplus b_7=0$ ②

$b_4\oplus b_5\oplus b_6\oplus b_7=0$ ③

$\oplus$表示逻辑加。

在一般情况下，校验码会被插入到数据的 1，2，4，8，…位置，那么，在数据生成时，按照提供的海明校验方程计算出 b_1，b_2，b_4，…各位，在数据校验时，按照海明检验方程进行计算，如果所有的方程式计算都为 0，则表示数据是正确的。如果出现 1 位错误，则至少有一个方程不为 0。海明码的特殊之处在于，只要将①②③ 3 个方程左边计算数据按③②①排列，得到的二进制数值就是该数据中出错的位，例如，第 6 位出错，则③②①为 110 等于二进制的 6。

当出现两位错误时，这种海明码能够查错，但无法纠错。

7.1.7 循环冗余校验码（CRC）

这种方式已经被广泛地在网络通信及磁盘存储时采用，所以在历年考试中出现的概率也比较大。先看几个基本概念。

1．多项式

在循环冗余校验码中，无一例外地要提到多项式的概念。一个二进制数可以以一个多项式来表示。如 1011 表示为多项式 $x^3+x^1+x^0$，在这里，x 并不表示未知数这个概念，如果把这里的 x 替换为 2，这个多项式的值就是该数的值。从这个转换可以看出多项式最高幂次为 n，则转换为二进制数有 $n+1$ 位。

2．编码的组成

循环冗余校验码校验由 k 位信息码，加上 R 位的校验码。

3．生成多项式

和海明码的校验方程一样，生成多项式非常重要，以至于考试中总是直接给出。

由 K 位信息码如何生成 R 位的校验码的关键在于生成多项式。这个多项式是编码方程和解码方程共同约定的，编码方程将信息码的多项式除以生成多项式，将得到余数多项式作为校验码，解码方程将收到的信息除以生成多项式，如果余数为 0，则认为没有错误。如果不为 0，余数则作为确定错误位置的依据。

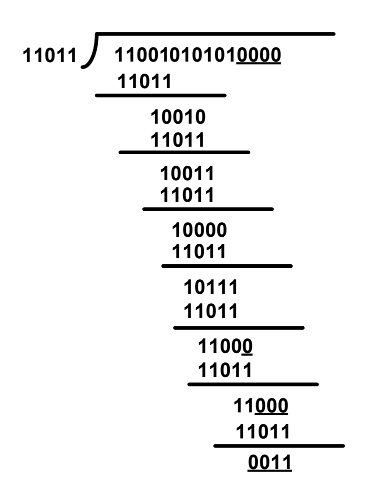
生成多项式并非任意指定，它必须具备以下条件：最高位和最低位为 1、2。数据发生错误时，余数不为 0，对余数补 0 后，继续做按位除，余数循环出现，这也是冗余循环校验中循环一词的来源。

4．校验码的生成

- 将k位数据$C(x)$左移R位，给校验位留下空间，得到移位后的多项式：$C(x)\times x^R$。
- 将移位后的信息多项式除以生成多项式，得到R位的余数多项式。
- 将余数嵌入信息位左移后的空间。

例如，信息位为 11001010101　　　生成多项式：$a(x)=x^4+x^3+x+1$（即 11011）

将信息位后补充 4 个 0，与生成多项式做模 2 除法，有：



得到余数为 0011，所以 CRC 码是：110010101010011。

循环冗余校验码的纠错能力取决于 k 值和 R 值。在实践中，k 取值往往取得非常大，远远大于 R 的值，提高了编码效率。在这种情况下，循环冗余校验就只能检错不能纠错。一般来说，R 位生成多项式可检测出所有双错、奇数位错和突发位错小于等于 R 的突发错误。使用循环冗余校验码能用很少的校验码检测出大多数的错误，检错能力非常强，这使得它得到了广泛的应用。

7.2　中央处理器（CPU）

现有的计算（包括单片机、PC、超级计算机）基本都是冯 • 诺依曼结构，这种结构将计算机分解成运算器、控制器、存储器、输入/输出设备，不加区别地将指令和数据存储在存储器中，指令、数据、存储地址都以二进制数表示，计算机运行时，执行的是存储器中的指令。由程序计数器来控制指令的执行。

中央处理器是计算机的控制、运算中心，它主要通过总线和其他设备进行联系，另外，在嵌入系统设计中，外部设备也常常直接接到中央处理器的外部I/0脚的中断脚上。

中央处理器的类型和品种异常丰富，各种中央处理器的性能差别很大，有不同的内部结构、不同的指令系统。但由于基于冯·诺依曼结构，基本部分组成相似。

1．运算器（ALU）

运算器的主要功能是在控制器的控制下完成各种算术运算、逻辑运算和其他操作。一个计算过程需要用到加法器/累加器、程序状态寄存器、其他数据寄存器等。

- 加法器/累加器（Accumulator）：专门存放算术或逻辑运算的一个操作数和运算结果的寄存器。能进行加、减、读出、移位、循环移位和求补等操作。是运算器的主要部分。
- 程序状态寄存器（Program Status Word）：是计算机系统的核心部件——运算器的一部分，状态寄存器用来存放两类信息，一类是体现当前指令执行结果的各种状态信息（条件码），如有无进位（CF位）、有无溢出（OV位）、结果正负（SF位）、结果是否为零（ZF位）、奇偶标志位（P位）等；另一类是存放控制信息，如允许中断（IF位）、跟踪标志（TF位）等。

2．控制器

控制器是中央处理器的核心，它控制和协调整个计算机的动作，控制通常需要程序计数器（PC），指令寄存器（IR），指令译码器（ID），定时与控制电路，以及脉冲源、中断等共同完成。

- 程序计数器（Program Counter）：程序计数器中存放的是下一条指令的地址。由于多数情况下程序是顺序执行的，所以程序计数器设计成能自动加1的装置。当出现转移指令时，就需重填程序计数器。
- 指令寄存器（Instruction Register）：显然，中央处理器即将执行的操作码表在这里。
- 指令译码器（Instruction Decoder）：将操作码解码，告诉中央处理器该做什么。
- 定时与控制电路（Programmable Logic Array）：用来产生各种微操作控制信号。

程序计数器可能是下一条指令的绝对地址，也可能是相对地址，即地址偏移量。

- 堆栈和堆栈指针（Stack Pointer）：堆栈可以是一组寄存器或在存储器内的特定区域。由于寄存器数量总是有限，所以大多数系统采用了使用存储器的软件堆栈。堆栈顶部的指针称为堆栈指针。

7.3 输入/输出控制方式

计算机与外设之间的数据交换被称为输入/输出，其控制方式主要有5种：程序查询方式、中断方式、DMA（Direct Memory Access）方式、通道方式、输入/输出处理机方式。由于通道方式与输入/输出处理机方式都已采用专用设备来管控输入/输出，所以在此不展开讨论。

1．程序查询方式

这是最简单的方式，也是简单系统中（外设种类和数目有限，数据传输速度较低的系统）常用的方式。这种方式是中央处理器定时查询外设的状态，如果发现某个外设就绪，就开始和该外设进行输入/输出操作和处理，如图7-1所示。

当存在多个外设时，中央处理器有串行和并行两种查询方式。串行查询是每次查询一个外设，并行则是将多个外设的状态位集中成一个专用端口，这样中央处理器一次查询即可得到多个外设的状态。

程序查询方式的缺点是：在输入/输出控制器和外设交换数据的过程中，中央处理器必须等待。这种等待对于许多系统而言是无法容忍的。

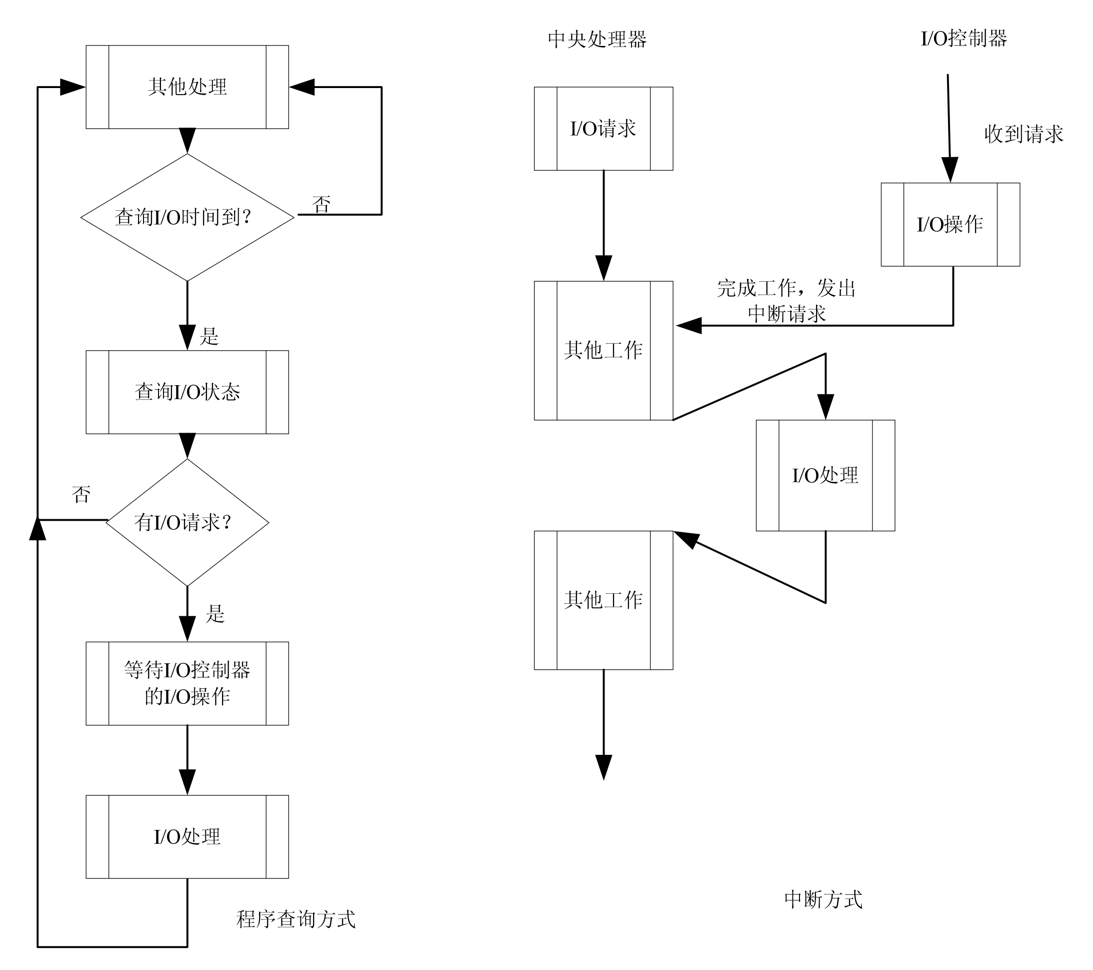


图 7-1　程序查询和中断方式

2．中断方式

使用中断方式，可以克服查询方式的低效问题。

当中央处理器执行到 I/0 请求指令时，向输入/输出控制器发出相应指令后，中央处理器并不等待，而是继续执行其他操作。此时，输入/输出控制器负责和外设进行通信，当数据从其数据寄存器写到外设后或者外设的数据写入其数据寄存器后，输入/输出控制器向中央处理器发出中断请求，中央处理器响应中断，并进行相应的处理。注意到，由于输入/输出控制器的数据寄存器大小有限，一次输入/输出请求往往要经过多次的中断过程才能够完成。由于中央处理器无须等待输入/输出控制器和外设的数据交换，从而提高了整个系统的效率。中断方式已经得到了普遍的应用。

1）中断的基本概念

中断并不只用于输入/输出系统中，中断系统是计算机的基本结构，中断系统的出现，是现代计算机功能强大的标志。顾名思义，中断就是打断中央处理器正在执行的工作，让中央处理器去处理其他更加重要或者更为紧迫的任务。发起中断的事务称为中断源，中断源包括 I/O 设备、实时时钟、故障源、软件中断等。中断系统使得中央处理器摆脱了只能按照指令顺序执行的束缚，让计算在并行性、分时操作、故障处理等方面更加强大。

按照中断源来区分中断，可以分为内部中断和外部中断。内部中断是中央处理器内部产生的中断，在个人计算机中，内部中断又分为溢出中断、除法错中断、断点中断、软中断及单步中断，其中可以使用软件中断实现 DOS 功能调用和基本 BIOS 调用，也可以使用单步中断实现程序的调试，与之相对应的是外部中断，中断源来自于中央处理器之外。而外部中断按照中央处理器的响应可以分为可屏蔽中断和非屏蔽中断。非屏蔽中断是中央处理器一定要响应的中断，通常是计算机发生了紧急情况，如掉电等。可屏蔽中断大多数是外设和时钟中断，在计算机处理一些不应该打断的任务时，可以通过屏蔽位来禁止响应这些中断。

2）中断处理过程

中央处理器收到中断请求后，如果是当前允许的中断，那么要停止正在执行的代码，并把内部寄存器入栈，这个过程不能被再次打断，所以在保护现场的开始要先关中断，保护完后再开中断。这个过程应该尽量短，以避免错过了其他中断。这个过程消耗的时间称为中断响应时间。然后开始执行中断处理程序，中断处理程序常常比较简单，通常是设置一些标志位，做一些简单的数据处理，而让其他更耗时的处理在非中断程序中完成。中断处理程序完成后，需要将刚才保存的现场恢复，把入栈的寄存器出栈，继续执行被中断的程序。整个过程消耗的时间称为中断处理时间，当然对于这个时间，不同的中断，不同的应用差别比较大，而且也不是一味求短，实际编写时要考虑中断处理的重要程度。现在大多数中央处理器都支持多级中断，即在进行中断处理程序时，还可以响应其他中断，形成中断嵌套。

3）中断的判断

当有多个中断源时，常用的处理方式有如下几种。

- 每个中断源使用自己的中断请求信号线和中央处理器相连，这种方式适用于中断源不是很多的情况，而中央处理器的外部中断引脚是有限的。
- 统一的中断请求，由中央处理器使用专门程序依次判断是哪个中断源的请求，通过查询的次序，可以实现中断的优先级控制。
- 硬件查询法：有一条中断确认信号链和输入/输出设备相连，某个外设发出中断请求后，中断确认信号开始在各外设间传递，发出中断请求的外设响应这个信号，如图7-2所示。

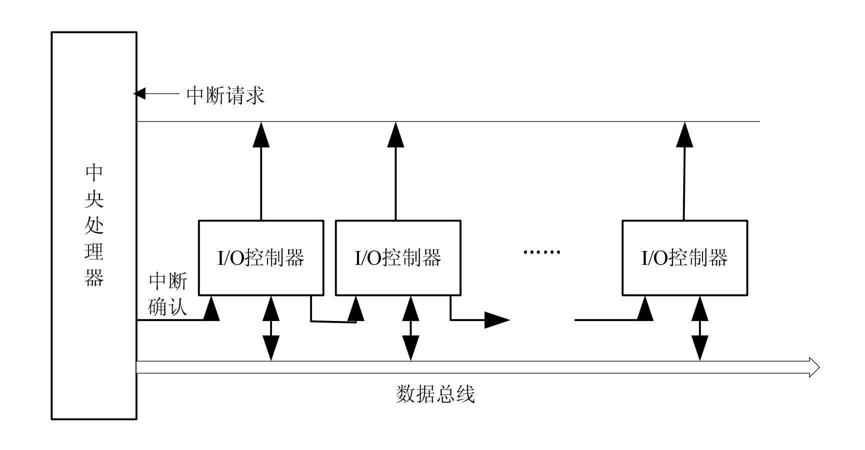


图 7-2　中断方式的判断

- 总线仲裁：在这种方式中，外设须先得到总线控制权，发出中断请求，最后将设备号通过数据总线发给中央处理器。由总线仲裁机制决定可以发信号的外设。
- 中断向量表：是一张不同中断处理程序入口地址的表格。用这种机制，每个中断源有不同的“中断号”，即中断向量中央处理器收到中断信号，并根据中断号查中断向量表，以得到该中断的处理程序的入口地址。

3．DMA 方式

DMA 直接存储器存取。这种方式可以使得数据从输入/输出模块到主存的传输过程中，无须中央处理器的中转，这个工作转移给了 DMA 控制器（DMAC）来完成，这种方式可以达到高速的数据传输。

1）DMAC 控制器

DMAC 也能访问系统总线，能够独立访问主存（这两个特点使得 DMAC 完成主存和输入/输出设备之间的数据交换），如图 7-3 所示。

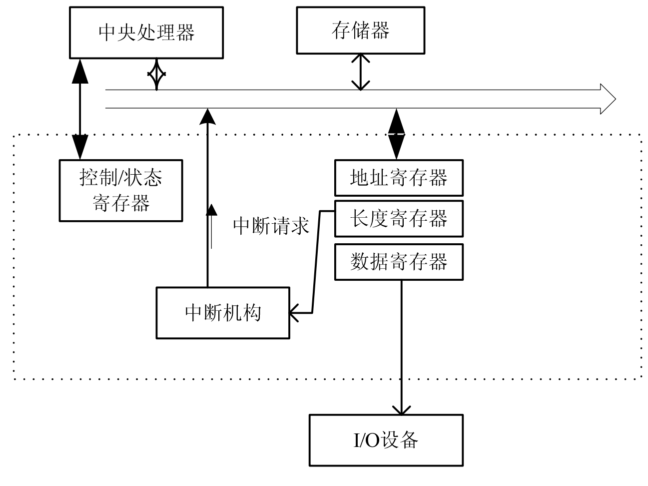


图 7-3　DMAC 示意图

DMA 中断控制示意图如图 7-4 所示。

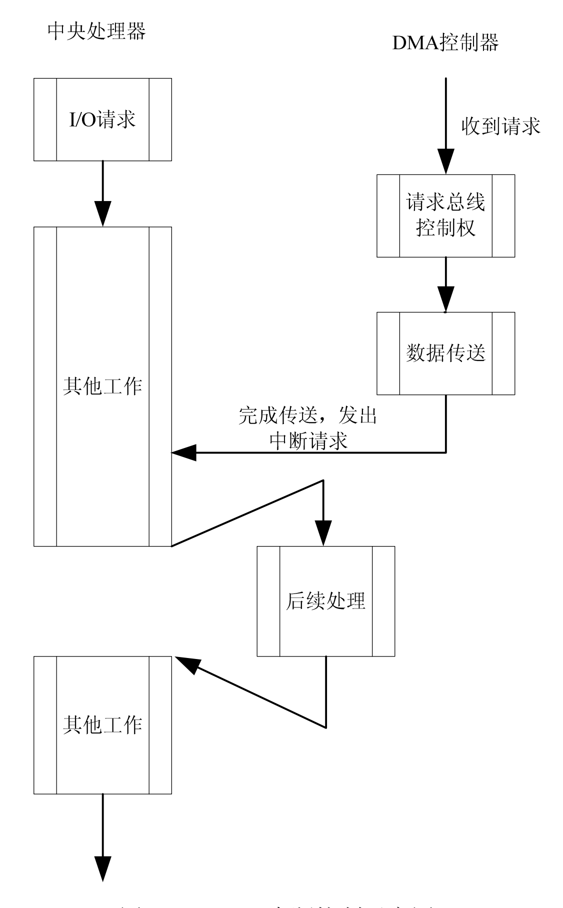


图 7-4　DMA 中断控制示意图

地址寄存器是内存地址，每传递一个数据，将这个寄存器加 1，长度寄存器值减 1，当长度寄存器为 0 时，发给中断机构完成信号，通知中央处理器进行后续处理。

当中央处理器执行到输入/输出请求时，向 DMA 控制器发出相应指令，DMA 控制器首先判断外设是否可用，如果可用，填充地址寄存器、长度寄存器等，向中央处理器发出总线请求信号，申请总线的处理权。中央处理器收到总线请求信号，让出总线控制权，然后 DMA 控制器将数据在外设和内存指定区域之间进行传送，而长度寄存器保存的值随着数据的传送不断减少，当减少到 0 时，通过中断机构向中央处理器发出中断请求，中央处理器响应中断，对内存中的数据进行后续的处理。

2）DMA 传送过程的总线占有方式

在 DMA 传输过程中，中央处理器停止访问主存，只进行了一些与总线无关的内部操作。这种方法常用于高速的输入/输出设备。

优点是减少系统总线控制权的交换次数，实现简单；缺点在于这样的结果往往使中央处理器在 DMA 过程中无所事事。

时间轮转片法：这种方法按照一定时间间隔，将总控制权分别轮换着交给中央处理器和 DMA。这样中央处理器不会停止工作，但往往外设的速度低，可能使得 DMAC 的某些时间空转。从效率而言，仍然不高。

借用周期法：这是时间轮转片的改进，即当有 DMA 操作时，DMAC 控制总线访问内存，其他时间总线的控制权在中央处理器，还适合于外设速度远低于总线速度的高速主机。这种方式由于要判断 DMAC 是否需要使用总线，所以实现起来要比前面两者要复杂。

3）DMA 方式和中断方式的区别

DMA 方式中用到了中断，但是 DMA 和中断的输入/输出方式是有很大区别的。最根本的区别在于，使用中断方式时，主存和输入/输出控制器之间的数据传送仍然需要用中央处理器来操作，需要使用中央处理器的寄存器等资源，如图 7-5 所示。

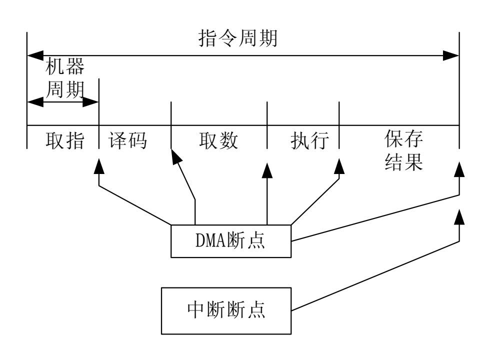


图 7-5　DMA 与中断方式的区别

而且，由于输入/输出控制器的数据寄存器大小有限，所以一个数据传送过程往往需要多次，这样中断发生就会很频繁。由于中断调用过程使用了中央处理器的资源，所以中央处理器必须保护现场，使得在相当程度上增加了处理时间。而在 DMA 传送过程中，虽然 DMA 控制器需要暂停中央处理器的执行，来达到控制总线的目的，但这种暂停是机器周期的中断，而且这个暂停中央处理器不需要保护现场，没有切换任务的操作。当数据传送完成后，才有一个中断，通知中央处理器进行数据传送的后续工作。DMA 方式提供了比中断方式更好的并行性，如表 7-2 所示。

表 7-2　DMA 方式和中断方式的比较

	中 断 方 式	DMA 方式
I/O 和主存数据传送	需要 CPU 处理	不需要 CPU 处理
保护执行现场	需要	不需要
相应时间	一条指令结束	CPU 周期结束
并行性	有	更好的并行性
处理异常能力	强	比中断差

7.4　指令流和数据流

指令流：机器执行的指令序列。

数据流：由指令流调用的数据序列，包括输入数据和中间结果。

按照计算机同时处于一个执行阶段的指令或数据的最大可能个数，将计算机分成 4 种，如表 7-3 所示。

表 7-3　指令流的分类

数据流 \ 指令流	单（Single）	多（Multiple）
单（Single）	SISD	MISD
多（Multiple）	SIMD	MIMD

SISD：这是最简单的方式，计算机每次处理一条指令，并只对一个操作部件分配数据。一般认为流水线技术的计算机仍然属于 SISD。

SIMD：具备 SIMD 点的常常是并行处理机，这种处理机具备多个处理单元，每次都执行同样的指令，对不同的数据单元进行处理。这种计算机非常适合处理矩阵计算等。

MISD：这种处理方式比较难以想象，有多个处理单元，同时执行不同的指令，针对的是单一数据。所以这种结构只是一种理论上的说法，并无实际应用。

MIMD：这是一种全面的并行处理机，典型的机型是多处理机。这种计算机的设计和控制都很复杂。

7.5　流水线技术

本节将介绍流水线技术。

7.5.1　流水线

还记得美国人泰勒吗？他有一个了不起的发现，即工人的机械劳动可细分为若干个环节。这样，如果所有工人都遵循固定的、优化了的劳动程序进行劳动，劳动效率就大为提高。资本家们进一步发挥，发展到每个工人只执行全部劳动细节的一小部分，一个产品由多个工人共同完成，这使得劳动效率飞升，这种技术称为流水线。

在中央处理器处理指令时，为提高效率，人们也采用了这种技术，对中央处理器而言，这实际是一种以硬件增加来换取性能提升的方式，由于硬件成本的持续下降，越来越多的中央处理器采用了这个技术：把一个指令分解成多个更小的指令，由不同的处理单元来处理，这样就形成了流水线。在理想的满负荷运行的状态下，执行一条指令的时间虽然没有减少，但是多个处理单元同时工作，在同一时间可以执行不同的指令的不同部分，从而使得总体的执行时间大大减少，减少到最慢的那一步的时间，如果各步骤处理时间相等，则一条指令分解成多少步，则处理速度就能提高多少倍，如图 7-6 所示。

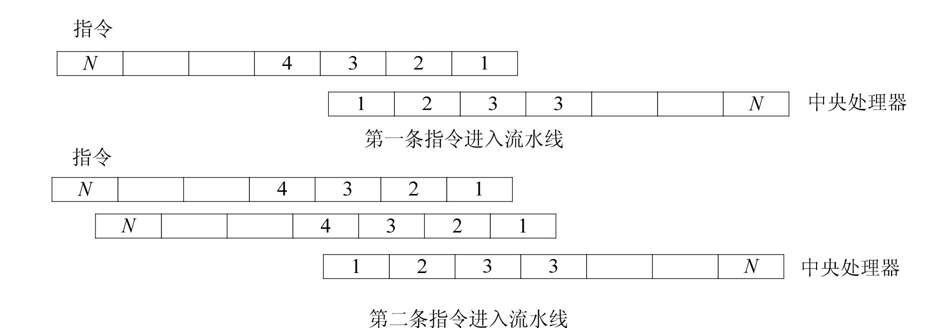


图 7-6　流水线的进行方式

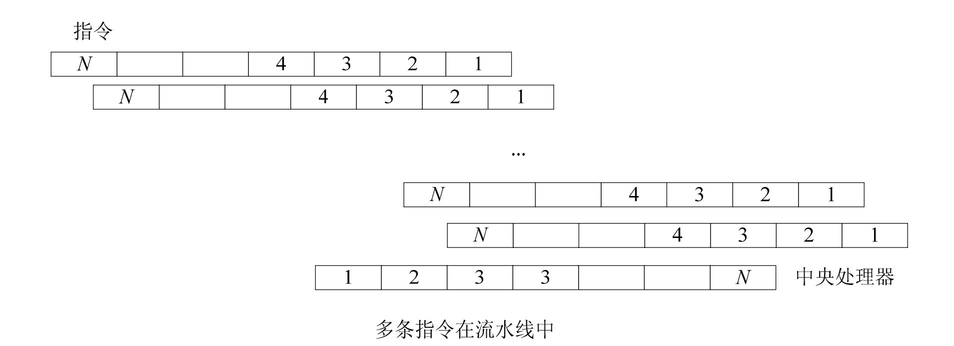


图 7-6 流水线的进行方式（续图）

例如，如果一个中央处理器把执行指令分成 7 步，如表 7-4 所示。

表 7-4 执行指令的七大步骤

步 骤	简 述	CPU 周期数
1.PC（程序计算器）	PC 自动加 1	1
2.取址	从 PC 指示的地址中取得完整指令	4
3.译码	可是硬件译码，也可能是微程序译码	1
4.取操作数地址	如果操作数是立即数可省略	1
5.取操作数		4
6.执行		1
7.保存结果		4

这种情况下，理想的满负荷执行的多个指令执行时间为 4 个 CPU 同期。

更进一步讲，中央处理器进行运算操作也可以利用流水线计数，例如，一个浮点加法运算，通常分解为 3 个阶段，如图 7-7 所示。

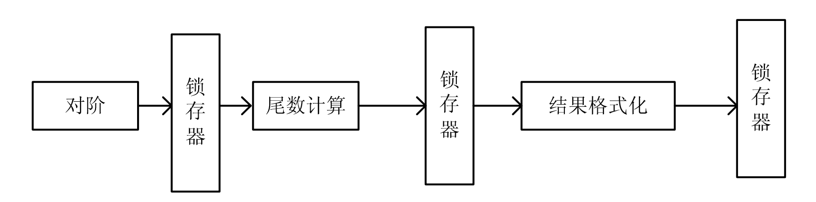


图 7-7 浮点运算的分解

锁存器的作用是在各步骤之间保存中间结果，这样理想状态下中央处理器把浮点加法的运算速度提高 3 倍。

前面多次提到理想状态和负荷两个词，显然在中央处理器流水线工作时，后面部分的处理单元处于无所事事的等待状态，只有在所有的工作单元都开始工作时，流水线才处于负荷的状态下。

理想状态是指没有阻塞的情况，现实中的流水线如某一环节出了问题，流水线的速度就会大为降低，中央处理器也是这样，那么影响流水线的因素有哪些呢？

7.5.2　影响流水线效率的因素

1．条件转移指令

最常见的就是条件转移指令，在存在转移指令的情况下，下一条需要执行的指令未必是程序计数器所指定的指令。只有在这条转移指令执行完成后，才能判断下一条指令是什么。

如果在遇到转移指令时，关闭流水线的进入端口，防止错误发生，这种方法无疑降低流水线的效率，而且程序中的条件转移是大量存在的，这势必使得流水线在很多时间内闲置，影响计算机的性能。

有的计算机采用猜测法，当发现条件转移指令时则系统猜测可能会跳转到的语句，如果猜测正确，则流水线正常运行；如果猜测错误，则需要清空当前流水线的内容，如图 7-8 所示。

还有一种需要编译系统的支持，如图 7-8 所示，方法是这样的，即将必须执行的 D 指令提前执行，在 D 指令执行之后，条件转移指令的结果出来后，再判断是 B 或者 C 进入流水线。同样也可以把 A 指令前的指令滞后到 A 指令执行后执行，这样能保持流水线闲置的时间尽可能短。但这必须是在 D 和 B、C 指令不存在前后依存关系的情况下采用。统计的结果还是不错的，50%的条件转移指令能够进行这样的优化。

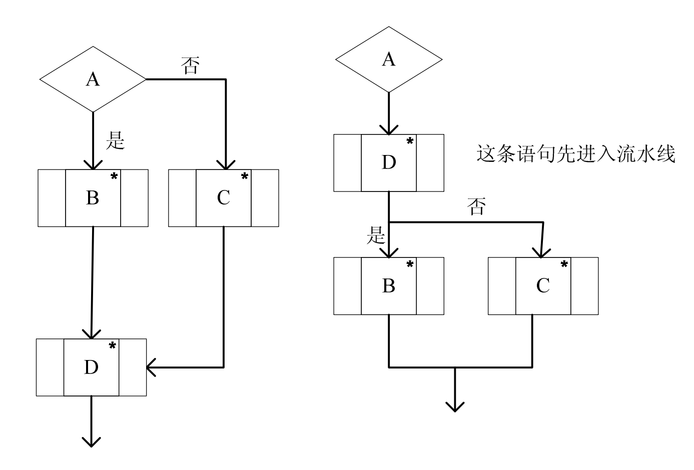


图 7-8　进入流水线的条件转移指令

2．资源共享

由于使用流水线，若相邻的两条指令都对同一个资源进行操作，或者前一条指令的输出是后一条指令的输入，在没有流水线情况下是正常的，在流水线时就可能出现错误。例如，前一条指令是写，后一条指令是读，当前一条指令保存结果没有完成时，后一条指令的读操作数就已经开始，这样后一条指令读到的就是未改写的数据。

为了解决这个问题，当遇到资源冲突时，就只好暂停后读指令进入流水线，这样会降低流水线的效率，显然，流水线步骤越多越容易引起资源冲突的发生。

也可以在编译系统上作文章，当发现相邻的语句存在资源共享冲突时，在两者之间插入其他语句，将两条指令进入流水线的时间拉开，以避免错误。

3. 寄存器相关

如果相邻的指令使用了相同的寄存器，这也会使得流水线失常。

通常的解决方法是如果此时还有其他寄存器可用，则给两个指令分配不同的寄存器，以避免冲突的发生。这对拥有大量的通用寄存器的精简指令系统计算机是个不错的方法。

4. 中断系统

当有中断发生时，和条件转移指令类似，流水线也不得不停止，以载入中断处理程序，由于中断的其他方面的优点，这种影响对流水线而言，是不可避免的。

流水线响应中断有两种方式，一种是立即停止现有的流水线，称为精确断点法，这种方法能够立即响应中断，缩短了中断响应时间，但是增加了中央处理器的硬件复杂度。

还有一种是在中断时，在流水线内的指令继续执行，停止流水线的入口，当所有流水线内的指令介绍执行后，再执行中断处理指令。这种方式中断响应时间较长，这种方式称为不精确断点法，优点是实现控制简单。

7.6 精简指令计算机

本节将介绍精简指令计算机。

7.6.1 指令系统

指令系统是中央处理器所有指令的汇集，也是高级软件编制的基础。指令系统的选择和确定涉及很多方面，是个复杂的问题。

通常一个指令可分解为：

操作码	地址码

前半部分的操作码确定指令的类型，后面的地址码确定指令所要处理的数据，根据地址码的个数可以有四址指令、三址指令，甚至是 0 址指令。

根据指令的长度特点，一个指令系统可能是定长指令字结构，即指令系统中所有的指令的长度都相同，特点是控制简单。如果指令的长度不固定，复杂的指令长度较长，就是变长指令字结构，这个结构的指令很容易扩展，但是增加了硬件的复杂度。

根据地址码代表的地址类型，指令系统可以有如下几种。

1）立即寻址

地址码就是操作数，这种寻址方式不必再次访问内存去取操作数，当然，也无须修改操作数。

2）直接寻址

地址码就是主存内数据的绝对地址，不必做任何换算。不足之处在于寻址范围有限，地址码的位数限制了寻址空间，而计算机的发展趋势是计算机拥有越来越大的内存。如果使用变长指令结构，则该指令就会变得臃肿。

3）寄存器寻址

地址码的地址是寄存器的地址。和内存寻址比较而言，访问寄存器的速度非常快，所以使用寄存器寻址有非常快的速度，不足之处在于寄存器的数量有限。

4）间接寻址

地址码指向主存中的数据，这个数据仍然是一个地址，这种方式提高了寻址的灵活性，扩大了寻址的范围。但由于要多次读主存，速度大为降低。

5）寄存器间接寻址

地址码保存的是寄存器地址，相对应的寄存器中保存的是数据的地址，这样既快又有灵活性，是一种广泛使用的寻址方式。

此外更复杂的寻址方式还包括：变址寻址、基址寻址、页面寻址和相对寻址。

7.6.2 CISC 和 RISC

随着硬件成本的下降，人们倾向于向中央处理器加入越来越多、越来越复杂的指令，同时，为了兼容老产品，原来的指令也要保留。这样，整个指令系统就向着越来越大、越来越复杂的趋势发展。在计算机处理能力越来越强的同时，中央处理器的设计也越来越复杂，这无疑在大大增加了设计的同时，更增加了设计失误的可能性。

事物的另外一个方面在于，增加指令的复杂性和中央处理器功能的增加似乎不一定是成正比的，人们发现在许多方面存在一个称为 20%～80%的定律，即系统中 20%的部分发挥了 80%的作用，通过对 CISC 指令系统的研究，发现系统在 80%的时间里执行的是 20%的指令。

于是出现了精简指令设计思想。这种计算机的指令结构不追求全面和复杂，而是只实现那些经常被执行的指令，由于指令比复杂指令结构计算机少得多，所以称为精简指令计算机。

先看著名的公式：$P=I\times \mathrm{CPI}\times T$

式中 P——计算机执行程序所需要的时间；

I——机器指令数；

CPI——平均每条指令所需要的机器周期数；

T——每个机器周期的时间。

CISC 在指令数上占优，而 RISC 在 CPI 上有则快得多，这是两种结构的两个方向。从这个公式可以发现，在理论上两者都有优势，不能认为精简指令计算机就好，复杂指令计算机就不好，事实上这两种设计方法很难找到完全的界线，而且在实际的芯片中，这两种设计方法也有相互渗透的地方，如表 7-5 所示。

表 7-5　两者的简单对比表

	CISC	RISC
指令条数	多	只选取最常见的指令
指令复杂度	高	低
指令长度	变化	短、固定
指令执行同期	随指令变化大	大多在一个机器同期完成
指令格式	复杂	简单
寻址方式	多	极少
涉及访问主存指令	多	极少，大部分只有存两条指令

续表

	CISC	RISC
通用寄存器数量	一般	大量
译码方式	微程序控制	硬件电路
对编译系统要求	低	高

第 8 章 存储器系统

按照考试大纲要求，本章要求考生掌握主存——Cache 存储系统的工作原理、虚拟存储器的基本工作原理、多级存储体系的性能价格、RAID 类型和特性等方面的内容。有关虚拟存储器的内容，放在“操作系统”一章中进行讨论。

计算机采用多级存储器体系，以确保能够获得尽可能高的存取速率，同时保持较低的成本。存储器体系包括寄存器、Cache、主存储器、磁盘存储器、光盘存储器、磁带存储器等，这些存储器从前到后，价格逐渐降低，容量和访问时间则逐渐增加。

存储器中数据常用的存取方式有顺序存取、直接存取、随机存取和相联存取等 4 种。

- 顺序存取：存储器的数据以记录的形式进行组织。对数据的访问必须按特定的线性顺序进行。磁带存储器采用顺序存取的方式。
- 直接存取：与顺序存取相似，直接存取也使用一个共享的读写装置对所有的数据进行访问。但是每个数据块都拥有唯一的地址标识，读写装置可以直接移动到目的数据块的所在位置进行访问。存取时间也是可变的，磁盘存储器采用直接存取的方式。
- 随机存取：存储器的每一个可寻址单元都具有自己唯一的地址和读写装置，系统可以在相同的时间内对任意一个存储单元的数据进行访问，而与先前的访问序列无关。主存储器采用随机存取的方式。
- 相联存取：相联存取也是一种随机存取的形式，但是选择某一单元进行读写取决于其内容而不是其地址。与普通的随机存取方式一样，每个单元都有自己的读写装置，读写时间也是一个常数。使用相联存取方式，可以对所有的存储单元的特定位进行比较，选择符合条件的单元进行访问。为了提高地址映射的速度，Cache采取相联存取的方式。

存储器系统的性能主要由存取时间、存储器带宽、存储器周期和数据传输率等来衡量。

8.1 主存储器

主存储器也简称为主存或内存，根据工艺和技术不同，可分为如下几种。

- RAM（Random Access Memory）：RAM存储器既可以写入也可以读出，但断电后信息无法保存，因此只能用于暂存数据。RAM又可分为 DRAM和 SRAM两种。
- DRAM（Dynamic RAM）：信息会随时间逐渐消失，因此需要定时对其进行刷新，维持信息不丢失。

- SRAM（Static RAM）：在不断电的情况下信息能够一直保持而不会丢失。

DRAM 的密度大于 SRAM 且更加便宜，但 SRAM 速度快，电路简单（无须刷新电路），然而容量小，价格高。

- ROM（Read Only Memory）：只读存储器，信息已固化在存储器中。ROM出厂时其内容由厂家用掩膜技术（Mask）写好，只可读出，但无法改写。一般用于存放系统程序BIOS和用于微程序控制。
- PROM（Programmable ROM）：可编程 ROM，只能进行一次写入操作（与 ROM相同），但是可以在出厂后，由用户使用特殊电子设备进行写入。
- EPROM（Erasable PROM）：可擦除的 PROM，其中的内容既可以读出，也可以写入。但是在一次写操作之前必须用紫外线照射 15～20分钟以擦去所有信息，然后再写入，可以写多次。
- E^2PROM（Electrically EPROM）：电可擦除EPROM，与 EPROM相似，可以读出也可写入，而且在写操作之前，不需要把以前的内容先擦去。能够直接对寻址的字节或块进行修改，只不过写操作所需的时间远远大于读操作所需时间（每字节需几百ms），其集成度也较低。
- 闪速存储器（Flash Memory）：其性能介于 EPROM与 E^2PROM之间。与E^2PROM相似，可使用电信号进行删除操作。整块闪速存储器可以在数秒内删除，速度远快于EPROM；而且可以选择删除某一块而非整块芯片的内容，但还不能进行字节级别的删除操作。集成度与 EPROM相当，高于E^2PROM。闪速存储器有时也简称为闪存。
- 相联存储器（Content Addressable Memory，CAM）：CAM是一种特殊的存储器，是一种基于数据内容进行访问的存储设备 CAM。当对其写入数据时，CAM能够自动选择一个未用的空单元进行存储；当要读出数据时，不是给出其存储单元的地址，而是直接给出该数据或者该数据的一部分内容，CAM对所有的存储单元中的数据同时进行比较并标记符合条件的所有数据以供读取。由于比较是同时、并行进行的，所以这种基于数据内容进行读写的机制，其速度比基于地址进行读写的方式要快许多。

8.2 辅助存储器

辅助存储器用于存放当前不需要立即使用的信息，一旦需要，再和主机成批交换数据，是主存储器的后备，因此称为辅助存储器；它又是主机的外围设备，又称为“外存储器”。辅助存储器的最大特点是存储器容量大、可靠性高、价格低。常用的辅助存储器有磁带存储器、磁盘存储器和光盘存储器。

8.2.1 磁带存储器

磁带存储设备是一种顺序存取的设备，存取时间较长，但存储容量大，价格便宜，目前仍用于数据备份。磁带的内容由磁带机进行读写（最便宜也最慢）。按磁带机的读写方式主要可以分为两种，启停式和数据流。

8.2.2 磁盘存储器

磁盘上的数据都存放于磁道上。磁道就是磁盘上的一组同心圆，其宽度与磁头的宽度相同。为了避免减小干扰，磁道与磁道之间要保持一定的间隔（Inter-Track Gap），沿磁盘半径方向，单位长度内磁道的数目称为道密度（道/英寸，TPI），最外层为 0 道。

沿磁道方向，单位长度内存储二进制信息的个数称为位密度。为了简化电路设计，每个磁道存储的位数都是相同的，所以其位密度也随着从外向内而增加。磁盘的数据传输是以块为单位的，所以磁盘上的数据也以块的形式进行存放。这些块就称为扇区（Sector），每个磁道通常包括 10～100 个扇区。同样为了避免干扰，扇区之间也相互留有空隙（Inter－Sector Gap）。柱面是若干个磁盘组成的磁盘组，所有盘面上相同位置的磁道组称为一个柱面（每个柱面有 n 个磁道）；若每个磁盘有 m 个磁道，则该磁盘组共有 m 个柱面。

磁盘的存取时间包括寻道时间和等待时间。寻道时间（查找时间，Seek Time）为磁头移动到目标磁道所需的时间（Movable－Head Disk），对于固定磁头磁盘而言，无须移动磁头，只需选择目标磁道对应的磁头即可。等待时间为等待读写的扇区旋转到磁头下方所用的时间。一般选用磁道旋转一周所用时间的一半作为平均等待时间。寻道时间由磁盘机的性能决定，单位一般以 ms 计，而转速则有 5 400rpm、7 200rpm、10 000rpm 等。软盘转速较慢，一般只有 360rpm（因为磁头与盘面接触性读写）。

磁盘的数据传输速率是指磁头找到地址后，单位时间写入或读出的字节数。R=TB/T，其中，TB 为一个磁道上记录的字节数，T 为磁盘每转一圈所需的时间，R 为数据传输速率。

8.2.3 RAID 存储器

廉价磁盘冗余阵列（Redundant Array of Inexpensive Disks，RAID）技术旨在缩小日益扩大的 CPU 速度和磁盘存储器速度之间的差距。其策略是用多个较小的磁盘驱动器替换单一的大容量磁盘驱动器，同时合理地在多个磁盘上分布存放数据以支持同时从多个磁盘进行读写，从而改善系统的 I/O 性能。小容量驱动器阵列与大容量驱动器相比，具有成本低、功耗小、性能好等优势；低代价的编码容错方案在保持阵列的速度与容量优势的同时保证了极高的可靠性。同时也较容易扩展容量。但是由于允许多个磁头同时进行操作以提高 I/O 数据传输速度，因此不可避免地提高了出错的概率。

为了补偿可靠性方面的损失，RAID 使用存储的校验信息（Stored Parity Information）来从错误中恢复数据。最初，Inexpensive 一词主要针对当时另一种技术（Single Large Expensive Disk，SLED）而言，但随着技术的发展，SLED 已逐渐被淘汰，RAID 和 non-RAID 皆采用了类似的磁盘技术。因此，RAID 现在代表独立磁盘冗余阵列（Redundant Array of Independent Disks），用 Independent 来强调 RAID 技术所带来的性能改善和更高的可靠性。

RAID 机制可分为如下级别。

- RAID 0级（无冗余和无校验的数据分块）：具有最高的I/O性能和最高的磁盘空间利用率，易管理，但系统的故障率高，属于非冗余系统，主要应用于那些关注性能、容量和价格而不是可靠性的应用程序。
- RAID 1级（磁盘镜像阵列）：由磁盘对组成，每一个工作盘都有其对应的镜像盘，上面保存着与工作盘完全相同的数据拷贝，具有最高的安全性，但磁盘空间利用率只有50%。RAID 1主要用于存放系统软件、数据，以及其他重要文件。它提供了数据的实时备份，一旦发生故障，所有的关键数据即刻就可使用。
- RAID 2级（采用纠错海明码的磁盘阵列）：采用了海明码纠错技术，用户需增加校验盘来提供单纠错和双验错功能。对数据的访问涉及阵列中的每一个盘。大量数据传输时I/O性能较高，但不利于小批量数据传输，实际应用中很少使用。
- RAID 3和RAID 4级（采用奇偶校验码的磁盘阵列）：把奇偶校验码存放在一个独立

的校验盘上。如果有一个盘失效，其上的数据可以通过对其他盘上的数据进行异或运算得到。读数据很快，但因为写入数据时要计算校验位，速度较慢。

- RAID 5（无独立校验盘的奇偶校验码磁盘阵列）：与RAID 4类似，但没有独立的校验盘，校验信息分布在组内所有盘上，对于大、小批量数据读写性能都很好。RAID 4和RAID 5使用了独立存取（Independent Access）技术，阵列中每一个磁盘都相互独立地操作，所以I/O请求可以并行处理。所以，该技术非常适合于I/O请求率高的应用而不太适应于要求高数据传输率的应用。与其他方案类似，RAID 4、RAID 5也应用了数据分块技术，但块的尺寸相对大一点。
- RAID 6：即带有两种分布存储的奇偶校验码的独立磁盘结构，它是对RAID 5的扩展，主要用于要求数据绝对不能出错的场合，使用了两种奇偶校验值，所以需要N+2个磁盘。同时对控制器的设计变得十分复杂，写入速度也不好，用于计算奇偶校验值和验证数据正确性所花费的时间比较多，造成了不必要的负载，很少人用。
- RAID 10：即RAID 1+RAID 0，高可靠性与高效磁盘结构。它是一个带区结构加一个镜像结构，可以达到既高效又高速的目的。这种新结构的价格高，可扩充性不好。

8.2.4　光盘存储器

光盘存储器是利用激光束在记录表面存储信息，根据激光束的反射光来读出信息。光盘存储器主要有 CD、DVD、蓝光，其中蓝光存储容量最大，单碟容量可达 50GB。光盘存储器的主要优点是：存储量很大且盘片易于更换，而缺点是：存储速度比硬盘低一个数量级。

8.3　Cache 存储器

Cache（高速缓冲存储器）的功能是提高 CPU 数据输入/输出的速率，突破所谓的“冯 •诺依曼瓶颈”，即 CPU 与存储系统间数据传送带宽限制。高速存储器能以极高的速率进行数据的访问，但因其价格高昂，如果计算机的主存储器完全由这种高速存储器组成则会大大增加计算机的成本。通常在 CPU 和主存储器之间设置小容量的高速存储器 Cache。Cache 容量小但速度快，主存储器速度较低但容量大，通过优化调度算法，系统的性能会大大改善，其存储系统容量与主存相当，而访问速度近似 Cache。在计算机的存储系统体系中，Cache 是访问速度最快的层次。

使用 Cache 改善系统性能的依据是程序的局部性原理（有关此原理的详细情况，读者可阅读“操作系统”相关章节）。依据局部性原理，把主存储器中访问概率高的内容存放在 Cache 中，当 CPU 需要读取数据时就首先在 Cache 中查找是否有所需内容，如果有则直接从 Cache 中读取；若没有再从主存中读取该数据，然后同时送往 CPU 和 Cache。如果 CPU 需要访问的内容大多都能在 Cache 中找到（称为访问命中，hit），则可以大大提高系统性能。

如果以 p 代表对 Cache 的访问命中率，$t1$ 表示 Cache 的周期时间，$t2$ 表示主存储器周期时间，以读操作为例，使用“Cache+主存储器”的系统的平均周期为 $t3$，则：$t3=p\times t1+(1-p)\times t2$。其中，$(1-h)$ 又称为失效率（未命中率）。

系统的平均存储周期与命中率有很密切的关系，命中率的提高即使很小，也能导致性能上的较大改善。

当 CPU 发出访问请求后，存储器地址先被送到 Cache 控制器以确定所需数据是否已在 Cache 中，若命中则直接对 Cache 进行访问。这个过程称为 Cache 的地址映射。常见的映

射方法有直接映射、全相联映射和组相联映射。

当 Cache 存储器产生了一次访问未命中之后，相应的数据应同时读入 CPU 和 Cache。但是当 Cache 已存满数据后，新数据必须淘汰 Cache 中的某些旧数据。最常用的淘汰算法有随机淘汰法、先进先出法（FIFO）和近期最少使用淘汰法（LRU）。

因为需要保证缓存在 Cache 中的数据与主存中的内容一致，相对读操作而言，Cache 的写操作比较复杂，常用的有如下几种方法。

- 写直达（Write Through）：当要写Cache时，数据同时写回主存储器，有时也称为写通。
- 写回（Write Back）：CPU修改Cache的某一行后，相应的数据并不立即写入主存储器单元。而是当该行从Cache中被淘汰时，才把数据写回到主存储器中。
- 标记法：对Cache中的每一个数据设置一个有效位。当数据进入Cache后，有效位置为1；而当CPU要对该数据进行修改时，数据只需写入主存储器并同时将该有效位清0。当要从Cache中读取数据时需要测试其有效位：若为1则直接从Cache中取数，否则从主存中取数。

第 9 章

安全性、可靠性与系统性能评测

根据考试大纲，本章要求考生掌握如下知识点：

- 安全性基本概念。
- 防治计算机病毒、防范计算机犯罪。
- 加密与解密机制。
- 存取控制、防闯入、安全管理措施。
- 诊断与容错。
- 系统可靠性分析评价。
- 计算机系统性能评测方式。
- 风险分析、风险类型、抗风险措施和内部控制。

有关风险分析的内容，读者可参考本书有关软件工程的章节。

9.1 数据安全与保密

国际标准化委员会对计算机安全的定义提出如下建议：“为数据处理系统建立和采取的技术的、管理的安全保护措施，用来保护计算机硬件、软件、数据不因偶然的、恶意的原因而遭破坏、更改和泄露。”计算机系统的安全主要包括网络安全、操作系统安全和数据库安全三个方面。

网络安全技术层次结构图如图 9-1 所示，包括各种安全技术和安全协议，分别对应于 OSI 七层网络协议的某一层或某几层，其中数据加密是计算机安全中最重要的技术措施之一。

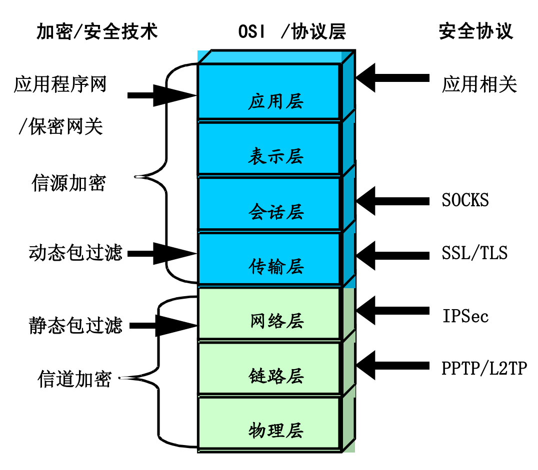


图 9-1 网络安全技术层次结构图

9.1.1 数据加密算法

数据加密是对明文（未经加密的数据）按照某种加密算法（数据的变换算法）进行处理，形成密文（经加密后的数据）。这样一来，密文即使被截获，截获方也无法或难以解码，从而防止泄露信息。

数据加密和数据解密是一对可逆的过程，数据加密是用加密算法 E 和加密密钥 $K1$ 将明文 P 变换成密文 C，表示为：

$$C = E_{K1}(P)$$

数据解密是数据加密的逆过程，用解密算法 D 和解密密钥 $K2$，将密文 C 转换成明文 P，表示为：

$$P = D_{K2}(C)$$

按照加密密钥 $K1$ 和解密密钥 $K2$ 的异同，有两种密钥体制。

- 秘密密钥加密体制 $K1=K2$：加密和解密采用相同的密钥，因而又称为对称密码体制。因为加密速度快，通常用来加密大批量的数据。典型的方法有DES算法、3DES算法、TDEA算法，Blowfish算法、RC5算法和IDEA算法。
- 公开密钥加密体制 $K1\neq K2$：又称为不对称密码体制，其加密和解密使用不同的密钥；其中一个密钥是公开的，另一个密钥则是保密的。由于加密速度较慢，所以往往用在数据量较小的通信业务中。典型的公开密钥加密方法有RSA、Elgamal、背包算法、Rabin、D-H、ECC（椭圆曲线加密算法）。

加密算法主要达到如下目的：提供高质量的数据保护，防止数据未经授权的泄露和未被察觉的修改；应具有相当高的复杂性，使得破译的开销超过可能获得的利益，同时又要便于理解和掌握；密码体制的安全性应该不依赖于算法的保密，其安全性仅以加密密钥的保密为基础；实现经济，运行有效，并且适用于多种完全不同的应用。

9.1.2 身份认证技术

主要以数字签名技术为例来说明。在某些商业或金融领域内，由于其行业要求，需要防止通信的一方否认或伪造通信内容，这时通常采用数字签名的方法。

数字签名用来保证信息传输过程中信息的完整和提供信息发送者的身份认证和不可抵赖性，该技术利用公开密钥算法对于电子信息进行数学变换，通过这一过程，数字签名存在于文档之中，不能被复制。该技术在具体工作时，首先发送方对信息施以数学变换，所得的变换信息与原信息唯一对应；在接收方进行逆变换，就能够得到原始信息。只要数学变换方法优良，变换后的信息在传输中就具有更强的安全性，很难被破译、篡改。这一过程称为加密，对应的反变换过程称为解密。

数字签名的算法很多，应用最为广泛的三种算法是：Hash 签名、DSS 签名和 RSA 签名。这三种算法可单独使用，也可综合在一起使用。

1．Hash 签名

Hash 签名不属于强计算密集型算法，应用较广泛。很多少量现金付款系统，如 DEC 的 Millicent 和 CyberCash 的 CyberCoin 等都使用 Hash 签名。使用较快的算法，可以降低服务器资源的消耗，减轻中央服务器的负荷。Hash 的主要局限是接收方必须持有用户密钥的副本以检验签名，因为双方都知道生成签名的密钥，较容易攻破，存在伪造签名的可能。

如果中央或用户计算机中有一个被攻破，那么其安全性就受到了威胁。

Hash 签名是最主要的数字签名方法，也称为数字摘要法、数字指纹法。它与 RSA 数字签名不同，该数字签名方法将数字签名与要发送的信息紧密联系在一起，它更适合于电子商务活动。将一个商务合同的个体内容与签名结合在一起，比合同和签名分开传递，更增加了可信度和安全性。

2．RSA 和 DSS 签名

RSA 和 DSS 都采用了公钥算法，不存在 Hash 的局限性。

RSA 是最流行的一种加密标准，许多产品的内核中都有 RSA 的软件和类库，早在 Internet 飞速发展之前，RSA 数据安全公司就负责数字签名软件与 Macintosh 操作系统的集成，在 Apple 的协作软件 PowerTalk 上还增加了拖放签名功能，用户只要把需要加密的数据拖到相应的图标上，即可完成电子形式的数字签名。RSA 与 Microsoft、IBM、Sun 和 Digital 都签订了许可协议，使在其生产线上加入了类似的签名特性。RSA 既可以用来加密数据，也可以用于身份认证。

用 RSA 或其他公开密钥密码算法进行数字签名的最大方便是没有密钥分配问题（网络越复杂、网络用户越多，其优点越明显）。因为公开密钥加密使用两个不同的密钥，其中有一个是公开的，另一个是保密的。公开密钥可以保存在系统目录内、未加密的电子邮件信息中、电话黄页（商业电话）上或公告牌里，网上的任何用户都可获得公开密钥。而保密密钥是用户专用的，由用户本身持有，它可以对由公开密钥加密信息进行解密。

RSA 算法中数字签名技术实际上是通过一个 Hash 函数来实现的。数字签名的特点是它代表了文件的特征，文件如果发生改变，数字签名的值也将发生变化。不同的文件将得到不同的数字签名。

DSS 是由美国国家标准化研究院和国家安全局共同开发的。由于该算法由美国政府颁布实施，主要用于与美国政府有商业往来的企业或组织，其他团体则较少使用。

DSS 的一个重要特点是两个素数公开，这样，当使用别人的 p 和 q 时，即使不知道私钥，我们也能确认它们是随机产生的，还是做了手脚。RSA 算法做不到这一点。

对数字签名和公开密钥加密技术来说，都会面临公开密钥的分发问题，即如何把一个用户的公钥以一种安全可靠的方式发送给需要的另一方。这就要求管理这些公钥的系统必须是值得信赖的。在这样的系统中，如果小王想要给老张发送一些加密数据，就需要知道老张的公开密钥；如果老张想要检验小王发来文档的数字签名，就需要知道小王的公开密钥。

所以，必须有一项技术来解决公钥与合法拥有者身份的绑定问题。假设有一个人自称某一个公钥是自己的，必须有一定的措施和技术来对其进行验证。

数字证书是解决这一问题的有效方法。它通常是一个签名文档，标记特定对象的公开密钥。数字证书由一个认证中心（CA）签发，认证中心类似于现实生活中公证人的角色，它具有权威性，是一个普遍可信的第三方。当通信双方都信任同一个 CA 时，两者就可以得到对方的公开密钥，从而能进行秘密通信、签名和检验。

CA 是 Certificate Authority 的缩写，是证书授权的意思。在电子商务系统中，所有实体的证书都是由证书授权中心即 CA 中心分发并签名的。一个完整、安全的电子商务系统必

须建立起一个完整、合理的CA体系。CA体系由证书审批部门和证书操作部门组成。

电子商务的安全是通过使用加密手段来达到的，公开密钥加密技术是电子商务系统中主要的加密技术，主要用于对称加密密钥的分发（数字信封）和数字签名，以实现身份认证和信息的完整性检验，以预防交易的抵赖等。CA体系为用户的公钥签发证书，以实现公钥的分发并证明其有效性。该证书证明了用户拥有证书中列出的公开密钥。证书是一个经证书授权中心签名的包含公开密钥拥有者信息以及公开密钥的文件。CA机构的数字签名使得攻击者不能伪造和篡改证书。证书的格式遵循X.509标准。

CA机构应包括两大部门：一是审核授权部门（Registry Authority，RA），它负责对证书申请者进行资格审查，决定是否同意给该申请者发放证书，并承担因审核错误引起的、为不满足资格证书申请者发放证书所引起的一切后果，因此它应由能够承担这些责任的机构担任；另一个是证书操作部门（Certificate Processor，CP），负责为已授权的申请者制作、发放和管理证书，并承担因操作运营所产生的一切后果，包括失密和为没有获得授权者发放证书等，它可以由审核授权部门自己担任，也可委托给第三方担任。

CA体系具有一定的层次结构，它由根CA、品牌CA、地方CA，以及持卡人CA、商家CA、支付网关CA等不同层次构成，上一级CA负责下一级CA数字证书的申请、签发及管理工作。通过一个完整的CA认证体系，可以有效地实现对数字证书的验证。每一份数字证书都与上一级的签名证书相关联，最终通过安全认证链追溯到一个已知的可信赖的机构。由此便可以对各级数字证书的有效性进行验证。根CA的密钥由一个自签证书分配，根证书的公开密钥对所有各方公开，它是CA体系中的最高层。

9.1.3 信息网络安全协议

目前，已经提出了大量的实用安全协议，有代表性的有：电子商务协议、IPSec协议、TLS协议、简单网络管理协议（SNMP）、PGP协议、PEM协议、S-HTTP协议、S/MIME协议等。对实用安全协议的安全性分析，特别是对电子商务协议、IPSec协议、TLS协议的分析是当前协议研究中的热点。典型的电子商务协议有SET协议、iKP协议等。另外，值得注意的是Kailar逻辑，它是目前分析电子商务协议的最有效的一种形式化方法。

为了实现安全IP，Internet工程任务组IETF于1994年开始了一项IP安全工程，专门成立了IP安全协议工作组IPSEC，来制定和推动一套称为IPSec的IP安全协议标准。其目标就是把安全集成到IP层，以便对Internet的安全业务提供低层的支持。IETF于1995年8月公布了一系列关于IPSec的建议标准。IPSec适用于IPv 4和下一代IP协议IPv 6，并且是IPv 6自身必备的安全机制。但由于IPSec还比较新，正处于研究发展和完善阶段。

在国际上，电子商务的安全机制正在走向成熟，并逐渐形成了一些国际规范，比较有代表性的有SSL协议和SET协议。

1．SSL协议

SSL（Security Socket Layer）协议是Netscape Communication 开发的传输层安全协议，用于在Internet上传送机密文件。该协议向基于TCP/IP的客户/服务器应用程序提供了客户端和服务器的鉴别、数据完整性及信息机密性等安全措施。该协议在应用程序进行数据交换前通过交换SSL初始握手信息来实现有关安全特性的审查。

SSL 首先要建立一条安全的连接，然后使用公钥加密方法传输数据。常用的浏览器（Netscape Navigator，Internet Explorer）都支持SSL，许多Web站点利用SSL获取用户的

机密信息，如信用卡号等。使用 SSL 连接的 URL，以 https://开头。另外一种在互联网上传送机密数据的协议是安全的超文本协议 S-HTTP（Security Hypertext Transfer Protocol）。SSK是在客户机和服务器之间建立一条安全的连接，而 S-HTTP 是安全地传送单个报文，属于应用层协议，因而这两个协议并非竞争的技术，而是互相补充的。SOCKS 是 IETF 的一个正式的标准，用于代理基于 TCP/IP 的网络应用。SOCKS 系统包含两个元素——SOCKS 服务器和 SOCKS 客户机。SOCKS 服务器实现于应用层，而 SOCKS 客户机实现于应用层和传输层之间。这个协议的主要作用是在两个没有直接 IP 联系的主机之间实现通信。

当客户机需要访问应用服务器时，客户机首先连接到 SOCKS 代理服务器上，代理服务器再连接到应用服务器上。代理服务器在客户机和应用服务器之间传送数据，如图 9-2 所示。对于应用服务器，代理服务器是客户机。

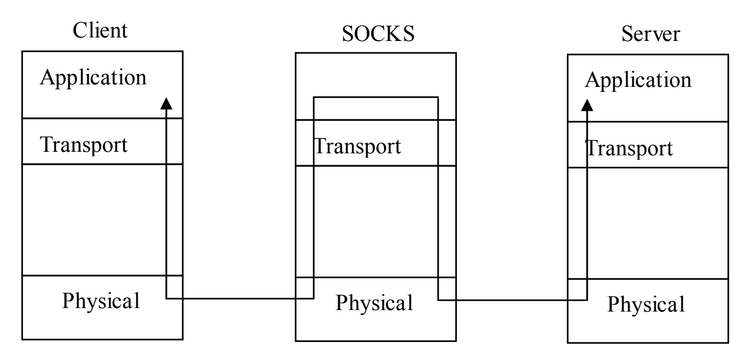


图 9-2　代理服务器传送数据图

2．SET 协议

SET（Secure Electronic Transaction）协议向基于信用卡进行电子化交易的应用提供了实现安全措施的规则。1995 年，信用卡国际组织、资讯业者及网络安全专业团体等开始组成策略联盟，共同研究开发电子商务的安全交易系统。1996 年 6 月，由 IBM、Master Card International、Visa International、Microsoft、Netscape、GTE、ViriSign、SAIC、Terisa 共同制定的标准 SET（Secure Electronic Transaction）正式公告，涵盖了信用卡在电子商务交易中的交易协定、信息保密、资料完整即数字认证、数字签名等。这一标准被公认为全球网络的标准，其交易形态将成为未来“电子商务”的典范。

SET 协议规定了交易各方进行安全交易的具体流程。在 SET 协议中，使用 DES 对称密钥算法、RSA 非对称密钥算法等提供数据加密、数字签名、数字信封等功能，给信息在网络中的传输提供了安全性保证。SET 协议通过 DES 算法和 RSA 算法的结合使用，保证了数据的一致性和完整性，并可实现交易以预防抵赖；通过数字信封、双重签章，确保用户信息的隐私性和关联性。

SET 协议执行步骤与常规的信用卡交易过程基本相同，只是它是通过因特网来实现的。在 SET 协议系统中，参与交易的主要有 4 种实体：持卡人、电子商家、收单银行、发卡银行。持卡人主要指持有信用卡的消费者；电子商家主要职能是支持网络购物的电子商店等提供电子交易服务的企业组织；收单银行主要使用支付系统的专用网关提供各商家的因特网在线借款服务；发卡银行负责处理信用卡的发放、账目管理、付款清算等。

此外，在协议系统中，还有认证中心，它是一些发卡机构共同委派的公证代理组织，其主要功能是产生、分配和管理发卡人、商家和参与电子交易等。

SET 协议规定的工作流程如下：用户向商家发送购货单和一份经过签名、加密的信托书。书中的信用卡号是经过加密的，商家无从得知；商家把信托书传送到收单银行，收单银行可以解密信用卡号，并通过认证验证签名；收单银行向发卡银行查问，确认用户信用卡是否属实；发卡银行认可并签证该笔交易；收单银行认可商家并签证此交易；商家向用户传送货物和收据；交易成功，商家向收单银行索款；收单银行按合同将货款划给商家；发卡银行向用户定期寄去信用卡消费账单。

SET 协议规定了参加电子交易各方在交易中的行为规范和信息交换的过程和规则，有助于实现安全、可靠的电子商务，得到了 IBM、VeriFone、HP、Microsoft、Netscape 等一些著名网络和计算机公司的支持。但是，SET 协议实施起来很复杂，因而在短期内推广 SET 协议还存在一定的困难。

9.1.4 防火墙技术

防火墙是位于两个（或多个）网络间，实施网络间访问控制的一组组件的集合，它是一套建立在内外网络边界上的过滤封锁机制。它满足如下条件：内部和外部之间的所有网络数据流必须经过防火墙，只有符合安全政策的数据流才能通过防火墙，防火墙自身应对渗透免疫。归纳起来，防火墙的功能有：过滤掉不安全服务和非法用户；控制对特殊站点的访问；提供了监视 Internet 安全和预警的方便端点。

设置防火墙的目的是为了保护内部不受来自 Internet 的攻击，为了创建安全域，为了增强一个机构内部网络的安全策略。防火墙需要满足两大需求：保障内部网络安全和保证内部网络同外部网的连通；通常内部网络被认为是安全和可信赖的，而外部网络（通常是 Internet）被认为是不安全和不可信赖的。

防火墙如果从实现方式上来分，分为硬件防火墙和软件防火墙两类。通常意义上讲的硬防火墙为硬件防火墙，它通过硬件和软件的结合来达到隔离内、外部网络的目的，价格较贵，但效果较好，一般小型企业和个人很难实现；软件防火墙是通过纯软件的方式来达到的，这类防火墙只能通过一定的规则来达到限制一些非法用户访问内部网的目的。

实现防火墙的产品主要两大类：一类是网络级防火墙，另一类是应用级防火墙。目前一种趋势是把这两种技术结合起来。

1）网络级防火墙

网络级防火墙也称为过滤型防火墙。事实上是一种具有特殊功能的路由器，采用报文动态过滤技术，能够动态地检查流过的 TCP/IP 报文或分组头，根据企业所定义的规则，决定禁止某些报文通过或者允许某些报文通过，允许通过的报文将按照路由表设定的路径进行信息转发。相应的防火墙软件工作在传输层与网络层。

状态检测防火墙又称为动态包过滤，是在传统包过滤上的功能扩展。状态检测防火墙在网络层由一个检查引擎截获数据包并抽取出与应用层状态有关的信息，并以此作为依据决定对该连接是接受还是拒绝。这种技术提供了高度安全的解决方案，同时也具有较好的性能、适应性和可扩展性。状态检测防火墙一般也包括一些代理级的服务，它们提供附加的对特定应用程序数据内容的支持。状态检测技术最适合提供对 UDP 协议的有限支持。它将所有通过防火墙的 UDP 分组均视为一个虚拟连接，当反向应答分组送达时，就认为一个

虚拟连接已经建立。状态检测防火墙克服了包过滤防火墙和应用代理服务器的局限性，不仅仅检测“to”或“from”的地址，而且也不要求每个访问的应用都有代理。

2）应用级防火墙

应用级防火墙也称为应用网关型防火墙，目前已大多采用代理服务机制，即采用一个网关来管理应用服务，在其上安装对应于每种服务的特殊代码（代理服务程序），在此网关上控制与监督各类应用层服务的网络连接。比如对外部用户（或内部用户）的FTP、TELNET、SMTP 等服务请求，检查用户的真实身份、请求合法性和源 IP 地址、目的地 IP 地址等，从而由网关决定接受或拒绝该服务请求，对于可接受的服务请求由代理服务机制连接内部网与外部网。代理服务程序的配置由企业网络管理员所控制。

目前常用的应用级防火墙大致有 4 种类型，分别适合于不同规模的企业内部网：双穴机网关、屏蔽主机网关、屏蔽子网网关和应用代理服务器。一个共同点是需要有一台主机（堡垒主机）来负责通信登记、信息转发和控制服务提供等任务。

- 双穴主机（Dual-Homed）网关：由堡垒主机作为应用网关，其中装有两块网卡分别连接外因特网和受保护的内部网，该主机运行防火墙软件，具有两个IP地址，并且能隔离内部主机与外部主机之间的所有可能连接。
- 屏蔽主机（Screened Host）网关：也称为甄别主机网关。在外部因特网与被保护的企业内部网之间插入了堡垒主机和路由器，通常是由IP分组过滤路由器去过滤或甄别出可能的不安全连接，再把所有授权的应用服务连接转向应用网关的代理服务机制。
- 屏蔽子网（Screened Subnet）网关：也称为甄别子网网关，适合于较大规模的网络使用。

即在外部因特网与被保护的企业内部网之间插入了一个独立的子网，比如在子网中有两个路由器和一台堡垒主机（其上运行防火墙软件作为应用网关），内部网与外部网的一方各有一个分组过滤路由器，可根据不同甄别规则接受或拒绝网络通信，子网中的堡垒主机（或其他可供共享的服务器资源）是外部网与内部网都可能访问的唯一系统。

- 应用代理服务器（Application Gateway Proxy）：在网络应用层提供授权检查及代理服务。当外部某台主机试图访问受保护网络时，必须先在防火墙上经过身份认证。通过身份认证后，防火墙运行一个专门为该网络设计的程序，将外部主机与内部主机连接。在这个过程中，防火墙可以限制用户访问的主机、访问时间及访问的方式。同样，受保护网络内部用户访问外部网时也需先登录到防火墙上，通过验证后，才可访问。

应用网关代理的优点是既可以隐藏内部 IP 地址，又可以给单个用户授权，即使攻击者盗用了一个合法的 IP 地址，也通不过严格的身份认证。因此应用网关比报文过滤具有更高的安全性。但是这种认证使得应用网关不透明，用户每次连接都要受到认证，这给用户带来许多不便。这种代理技术需要为每个应用写专门的程序。

9.2 容错技术

容错技术是提高系统可靠度及可用度的有效手段。

容错是指计算机系统在运行过程中发生一定的硬件故障或软件错误时仍能保持正常工作而不影响正确结果的一种性能或措施。具有容错能力的计算机称为容错计算机，容错采用冗余方法来消除故障影响。

提高计算机可靠性的技术可以分为避错技术和容错技术。后者主要运用冗余技术来抵消由于故障所引起的影响。冗余技术是计算机容错技术的基础，一般可分为如下几种类型。

- 硬件冗余：以检测或屏蔽故障为目的而增加一定硬件设备的方法。
- 软件冗余：为了检测或屏蔽软件中的差错而增加一些在正常运行时所不需要的软件方法。
- 信息冗余：除实现正常功能所需要的信息外，再添加一些信息，以保证运行结果正确，纠错码就是信息冗余的例子。
- 时间冗余：使用附加一定时间的方法来完成系统功能。这些附加的时间主要用在故障检测、复执或故障屏蔽上。

简单的双机备份：在 20 世纪 60 年代主要利用双处理机或双机的方法来达到容错的目的。例如，把关键的元件（处理机、存储器等）或整个计算机设置两套：一是系统运行时使用，另一份作为备份。根据系统的工作情况又可分为热备份和冷备份两种。

- 热备份（双重系统）：两套系统同时同步运行，当联机子系统检测到错误时，退出服务进行检修，而由热备份子系统接替工作。
- 冷备份（双工系统）：处于冷备份的子系统平时停机或者运行与联机系统无关的运算，当联机子系统产生故障时，人工或自动进行切换，使冷备份系统成为联机系统。在冷备份时，不能保证从程序端点处精确地连续工作，因为备份机不能取得原来的机器上当前运行的全部数据。

操作系统支持的双机容错：20 世纪在 70 年代中期出现了软件和硬件结构的容错方法。该方法在操作系统的层次上支持联机维修，即故障部分退出后运行、进行维修并重新投入运行都不影响正在运行的应用程序。该结构特点是系统内包括双处理器、双存储器、双输入/输出控制器、不间断工作的电源，以及与之适应的操作系统等。因此，上述硬件的责任一部分发生故障都不会影响系统的继续工作。系统容错是在操作系统控制下进行的，在每个处理机上都保持了反映所有系统资源状态的表格，以及本机和其他机器的工作进程。

9.3 系统可靠性评价和系统性能评价方法

本节将介绍系统可靠性评价和系统性能评价方法。

9.3.1 系统可靠性评价的组合模型

组合模型是计算容错系统可靠性最常用的方法。一个系统只要满足如下条件，就可以用组合模型来计算其可靠性。

- 系统只有两种状态：运行状态和失效状态。
- 系统可以划分成若干个不重叠的部件，每个部件也只有两种状态：运行状态和失效状态。
- 部件的失效是独立的。
- 系统失效当且仅当系统中的剩余资源不满足系统运行的最低资源要求（系统的状态只依赖于部件的状态）时。
- 已知每个部件的可靠性，可靠性指可用度或可靠度等概率参数。

常见的组合模型包括串联系统、并联系统、模冗余系统。

1）串联系统

假设一个系统由 n 个子系统组成，当且仅当所有的子系统都能正常工作时，系统才能正常工作，这种系统称为串联系统，如图 9-3 所示。

输入 R_1 R_2 …… R_n 输出

图 9-3　串联系统

设系统各个子系统的可靠性分别用 R_1，R_2，…，R_n 表示，则系统的可靠性 $R=R_1\times R_2\cdots\times R_n$。

如果系统的各个子系统的失效率分别用 λ_1，λ_2,…，λ_n 来表示，则系统的失效率 $\lambda=\lambda_1+\lambda_2+\cdots+\lambda_n$。

2）并联系统

假如一个系统由 n 个子系统组成，只要有一个子系统能够正常工作，系统就能正常工作，如图 9-4 所示。

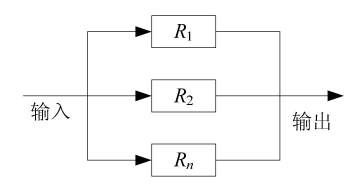


图 9-4　并联系统

设系统各个子系统的可靠性分别用 R_1，R_2，…，R_n 表示，则系统的可靠性 $R=1-(1-R_1)\times(1-R_2)\times\cdots\times(1-R_n)$。

假如所有子系统的失效率均为 λ，则系统的失效率为 μ：

$$\mu=\frac{1}{\frac{1}{\lambda}\sum_{j=1}^{n}\frac{1}{j}}$$

在并联系统中只有一个子系统是真正需要的，其余 $n-1$ 个子系统称为冗余子系统，随着冗余子系统数量的增加，系统的平均无故障时间也会增加。

3）模冗余系统

m 模冗余系统由 m 个（$m=2n+1$ 为奇数）相同的子系统和一个表决器组成，经过表决器表决后，m 个子系统中占多数相同结果的输出作为系统的输出，如图 9-5 所示。

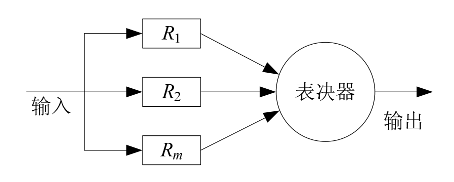


图 9-5　模冗余系统

在 m 个子系统中，只有 n+1 个或 n+1 个以上的子系统能正常工作，系统就能正常工作，输出正确结果。假设表决器是完全可靠的，每个子系统的可靠性为 R_0，则 m 模冗余系统的可靠性为：

$$\sum_{i=n+1}^{m} C_m^i R_0^i (1-R_0)^{m-i}$$

9.3.2　系统性能评价

计算机的性能主要反映了一个系统的使用价值，即性能价格比。性能的含义很广泛，主要包括系统处理能力（或吞吐率）、响应速度、可靠性、可使用性、可维护性等。这些性能既有定量的指标，又有定性的指标。

1．性能评价的常用指标及方法

1）时钟频率

计算机的时钟频率在一定程度上反映了机器速度，一般来说，主频越高，速度越快。但是相同频率、不同体系结构的机器，其速度可能会相差很多，因此，还需要用其他方法来测定机器性能。与时钟频率相关的另一个概念是性能因子，即 CPI 指每条指令的平均时钟周期。

$$\text{CPU}_{\text{time}} = \text{指令数} \times \text{CPI} \times \text{时钟周期} = \text{指令数} \times \text{CPI} / \text{时钟频率}$$

2）指令执行速度

在计算机发展的初期，曾用加法指令的运算速度来衡量计算机的速度，速度是计算机的主要性能指标之一。因为加法指令的运算速度大体上可反映出乘法、除法等其他算术运算的速度，而且逻辑运算、转移指令等简单指令的执行时间往往设计成与加法指令相同，因此加法指令的运算速度有一定代表性。表征机器运算速度的单位通常有每秒百万条指令（Million Instruction Per Second，MIPS）、每秒浮点指令数（Million Floating-point Instruction Per Second，MFLOPS）。

MIPS=指令数/（执行时间×1 000 000）

MIPS 大小和指令集有关，不同指令集的计算机间的 MIPS 不能比较；在同一台计算机上 MIPS 是变化的，因程序不同而变化；有时 MIPS 会出现矛盾，比如带有硬件浮点处理器的计算机；MIPS 中，除包含运算指令外，还包含取数、存数、转移等指令在内；MIPS 只适宜于评估标量机；相对 MIPS 是指相对参照机而言的 MIPS，通常用 VAX-11/780 机处理能力为 1MIPS。

MFLOPS=浮点指令数/（执行时间×1 000 000）

与机器和程序有关，测量浮点运算时，比 MIPS 准确；MFLOPS 比较适宜于评估向量计算机；MFLOPS 与 MIPS 之间的换算关系为：$1\text{MFLOPS} \approx 3\text{MIPS}$；MFLOPS 只能用来衡

量机器浮点操作的性能，而不能体现机器的整体性能。例如，编译程序，不管机器的性能有多好，它的 MFLOPS 不会太高；MFLOPS 是基于操作而非指令的，所以它可以用来比较两种不同的机器；单个程序的 MFLOPS 值并不能反映机器的性能；MFLOPS 依赖于操作类型，例如，100%的浮点加要远快于 100%的浮点除。

3）等效指令速度法

随着计算机指令系统的发展，指令的种类大大增加，用单位指令的 MIPS 值来表征机器的运算速度的局限性日益暴露，因此很快出现了改进的办法，称为吉普森（Gibson）混合法或等效指令速度法。

等效指令速度法用于统计各类指令在程序中所占的比例，并进行折算。设某类指令 i 在程序中所占比例为 w_i，执行时间为 t_i，则等效指令的执行时间为：

$$T = \sum (W_i \times t_i)$$

式中 n 为指令的种类数。

4）数据处理速率法

由于在不同程序上，各类指令的使用频率是不同的，所以固定比例方法存在着很大的局限性；而且数据长度与指令功能的强弱对解题的速度影响极大。同时这种方法也不能反映现代计算机中高速缓冲存储器、流水线、交叉存储等结构的影响。具有这种结构的计算机的性能不仅与指令的执行频率有关，而且也与指令的执行顺序与地址分布有关。

数据处理速率 PDR 法采用计算“数据处理速率”值的方法来衡量机器性能，PDR 值越大，机器性能越好。PDR 与每条指令和每个操作数的平均位数及每条指令的平均运算速度有关，其计算方法如下。

$$\text{PDR} = L / R$$

其中，$L = 0.85G + 0.15H + 0.4J + 0.15K$

$$R = 0.85M + 0.09N + 0.06P$$

式中，G 是每条定点指令的位数；M 是平均定点加法时间；H 是每条浮点指令位数；N 是平均浮点加法时间；J 是定点操作数的位数；P 是平均浮点乘法时间；K 是浮点操作数的位数。

此外，还做了如下规定：G＞20 位、H＞30 位；从主存取一条指令的时间等于取一个字的时间；指令与操作数存放在主存，无变址或间址操作；允许有并行或先行取址指令功能，此时选择平均取指令时间。PDR 值主要对 CPU 和主存储器的速度进行度量，但不适合衡量机器的整体速度，因为它没有涉及 Cache、多功能部件等技术性能的影响。

5）核心程序法

上述性能评价方法主要针对 CPU（有时包括主存），它没有考虑诸如 I/O 结构、操作系统、编译程序的效率等对系统性能的影响。因此，难以准确评价计算机的实际工作能力。

核心程序法是研究较多的一种方法，它把应用程序中用得最频繁的那部分核心程序作为评价计算机性能的标准程序，在不同的机器上运行，测得其执行时间，作为各类机器性能评价的依据。机器软/硬件结构的特点能在核心程序中得到反映，但是核心程序各个部分之间的联系较小。由于程序短，所以访问存储器的局部性特征很明显，以致 Cache 的命中

率比一般程序高。

基准程序法是目前一致承认的测试性能的较好方法，有多种多样的基准程序，如主要测试整数性能的基准程序，测试浮点性能的基准程序等。

2．基准测试程序

1）整数测试程序

Dhrystone 是一个综合性的基准测试程序，它是为了测试编译器和 CPU 处理整数指令和控制功能的有效性，人为地选择一些“典型指令”综合起来形成的测试程序。

用 C 语言编写的 Dhrystone 基准程序用了 100 条语句，由如下操作组成：各种赋值语句；各种数据类型的数据区；各种控制语句；过程调用和参数传送；整数运算和逻辑操作。

Dhrystone 程序测试的结果为每秒 1 757Dhrystones，为便于比较，人们假设 1 VAX MIPS=1 757Dhrystones 每秒，将被测机器的结果除以 1 757，就得到被测机器相对 VAX 11/780 的 MIPS 值。有些厂家在宣布机器性能时就用 Dhrystone MIPS 值作为各自机器的 MIPS 值。

不过不同厂家在测试 MIPS 值时，使用的基准程序一般不一样，因此不同厂家机器的 MIPS 值有时虽然相同，但性能却可能相差很大。这是因为各厂家在设计计算机时针对不同的应用领域：如科学和工程应用、商业管理应用、图形处理应用等，而采用了不同的体系结构和实现方法。同一个厂家的机器，采用相同的体系结构，用相同的基准程序测试，得到的 MIPS 值越大，一般说明机器速度越快。

2）浮点测试程序

在计算机科学工程应用领域内，浮点计算工作量占很大比例，因此机器的浮点性能对系统的应用有很大的影响。有些机器只标出单个浮点操作性能，如浮点加法、浮点乘法时间。而大部分工作站则用 Linpack 和 Whetstone 基准程序测得浮点性能。Linpack 主要测试向量性能和高速缓存性能。Whetstone 是一个综合性测试程序，除测试浮点操作外，还测试整数计算和功能调用等性能。

Linpack 基准测试程序：是一个用 Fortran 语言写成的子程序软件包，称为基本线性代数子程序包，此程序完成的主要操作是浮点加法和浮点乘法操作。测量计算机系统的 Linpack 性能时，让机器运行 Linpack 程序，测量运行时间，将结果用 MFLOPS 表示。

当解 n 阶线性代数方程组时，n 越大，向量化程度越高。其关系如表 9-1 所示。

表 9-1　n 与向量化的关系

矩阵规模	100×100	300×300	1 000×1 000
向量化百分比	80%	95%	98%

向量化百分比指的是向量成分的计算量占整个程序计算量的百分比。在同一台机器中，向量化程度越高，机器的运算速度越快，因为不管 n 的大小，求解方程时花费的非向量操作的时间差不多相等。

Whetstone 基准测试程序：Whetstone 是用 Fortran 语言编写的综合性测试程序，主要由执行浮点运算、整数算术运算、功能调用、数组变址、条件转移和超越函数的程序组成。Whetstone 的测试结果用 Kwips 表示，1Kwips 表示机器每秒钟能执行 1 000 条 Whetstone 指令。

3）SPEC 基准程序

SPEC 是 System Performance Evaluation Cooperative 的缩写，是几十家世界知名计算机大厂商所支持的非盈利的合作组织，旨在开发共同认可的标准基准程序。

SPEC 基准程序是由 SPEC 开发的一组用于计算机性能综合评价的程序。以对 VAX11/780 机的测试结果作为基数，其他计算机的测试结果以相对于这个基数的比率来表示。SPEC 基准程序能较全面地反映机器性能，具有一定的参考价值。

4）TPC 基准程序

事务处理委员会 Transaction Processing Council 简称为 TPC。TPC 基准程序是由 TPC 开发的评价计算机事务处理性能的测试程序，用以评价计算机在事务处理、数据库处理、企业管理与决策支持系统等方面的性能。TPC 成立于 1988 年，目前已有 40 多个成员，几乎包括所有主要的商用计算机系统和数据库系统。该基准程序的评测结果用每秒完成的事务处理数 TPC 来表示。TPC 基准测试程序在商业界范围内建立了用于衡量机器性能及性能价格比的标准。

第 10 章

网络基础知识

从古代的驿站、八百里快马，到近代的电报、电话，人类对于通信的追求从未间断过，信息的处理与通信技术的革新一直伴随社会的发展。

而作为 20 世纪人类最伟大、最卓越的发明，个人计算机的出现与发展使得人们获得了以前无法想象的信息处理能力，为了将这些强大的信息处理设备连接起来，避免出现“信息孤岛”现象，就催生了“计算机网络”这一新时代的通信技术。网络技术使得计算机的功能得到了大大的加强，使用范围得到了很大的扩展。

10.1　网络的功能、分类与组成

什么是计算机网络呢？计算机网络是指由通信线路互相连接的许多独立自主工作的计算机构成的资源共享集合体，它是计算机技术和通信技术相结合的产物。其中，通信线路并不专指铜导线，还可以是光纤，甚至可以是一些无界的媒体：如激光、微波、红外线等。在这个定义中，可以知道如下内容。

- 计算机网络的作用：资源共享。
- 计算机网络的组成：许多独立自主工作的计算机。
- 计算机网络的实现方式：使用通信线路互相连接。

另外，早期的计算机网络以一台或几台大型的计算机为中心，但由于计算机技术的十倍速发展，小型机甚至微型机都拥有了惊人的处理能力，而且在整体性能上均已超过了早期的大型计算机。所以网络的重心开始有了偏向，开始体现共享这一原则，也就是所有的计算机都具备了独立自主工作的能力。计算机网络从共享大型计算机的计算能力发展为共享存储在计算机内的信息，这也是时代发展所致。

10.1.1　计算机网络的分类

我们经常根据计算机网络的传输距离来进行分类，这是因为计算机间的距离、所要求的传输速度就决定了网络技术之间的差异。

不同传输距离的网络可以分为局域网、城域网、广域网三种。局域网的相关技术是基于处理近距离传输而设计和发展而来的，而广域网的相关技术是基于处理远距离传输而设计和发展而来的，城域网则为一个城市网络设计的相关技术。

10.1.2　按工作模式分类

根据在计算机网络中各个计算机所占的不同地位和所起的不同作用，以及它们的相互依赖情况，可以分为两种类型：对等网络和基于服务器的网络。这两种网络实现的复杂程度不同，性能不同，架设成本不同。

1．对等网络

所谓对等网络，指的是由一些直接面向用户的PC组成，这些PC是以对等的方式操作的，互相作为其他 PC 的资源来共享和使用。它们之间并没有主次之分，就像一组相互协作的伙伴。其结构如图10-1所示。

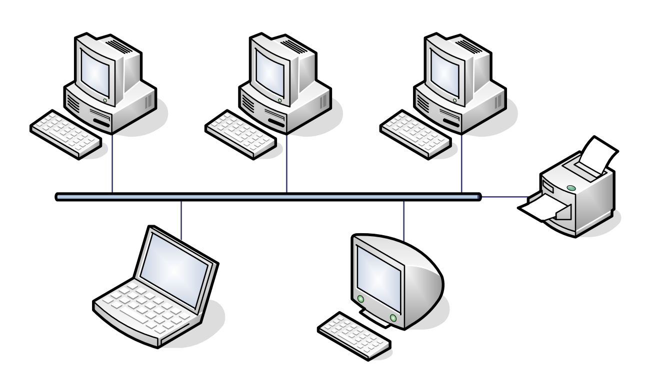


图10-1　对等网络结构示意图

对等网络的优点如下：

- 架设成本相对来说是十分廉价的。
- 建立、安装的过程简单，实现快捷。
- 由于结构简单，一般无须配备专门的网络管理人员。
- 用户对自己的资源能够完全管理，决定是否共享。

正是因为对等网络简单的实现，也存在着如下缺陷：

- 可扩充能力十分有限。
- 无法进行集中的管理，所以导致管理相对混乱。
- 正是因为各自为政，所以安全性不高。
- 在共享时有可能既成为“服务器”，又是“客户机”，所以负担较重。

一般情况下，对等网络是在联网计算机不超过10台，并且对安全、管理方面的要求不高的情况下，追求更高的性价比，这是最佳选择。切记，当节省下一笔开支的时候，同时也省下了不少“安全性”和“可管理性”。

2．基于服务器的网络

与对等网络完全不同，基于服务器的网络则是在一组 PC 中，包含专用的高性能计算机，这些计算机专门执行某些任务，比如文件服务器、打印服务器、数据库服务器、Web服务器、电子邮件服务器。它们之间是客户/服务器的关系，一个是提供服务，一个是使用服务，有鲜明的主次之分，以保证服务的可靠性。其结构如图10-2所示。

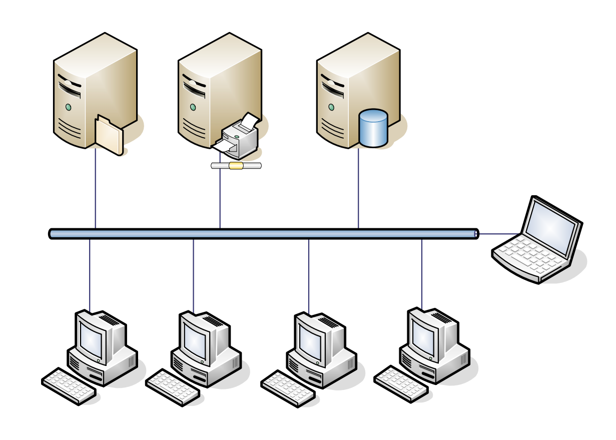


图 10-2　基于服务器的网络结构示意图

基于服务器网络的优点如下：

- 可以集中管理大量的用户，整个网络的可控制性较好。
- 这样的集中管理无形中提高了网络的安全性。
- 这种结构的可扩展性较好，有利于网络的扩大。
- 由于其结构分明，能够建立成冗余系统。

任何事物都是相对存在的，虽然基于服务器的网络有着这样那样的特点，但是仍然带来了新的问题：

- 专用的高性能服务器的使用当然也就提高了网络的整体造价。
- 集中管理用户和资源就使得网络的建立相对复杂。
- 正是因为网络更加复杂，所以通常需要有专门的网络管理人员来管理。

10.1.3　计算机网络的组成

总的来说，计算机网络由资源子网和通信子网两部分组成。

如图 10-3 所示，在虚线框内的就是负责信息传输的通信子网，而在框外进行数据通信和使用数据通信的主机、客户机都属于资源子网部分。

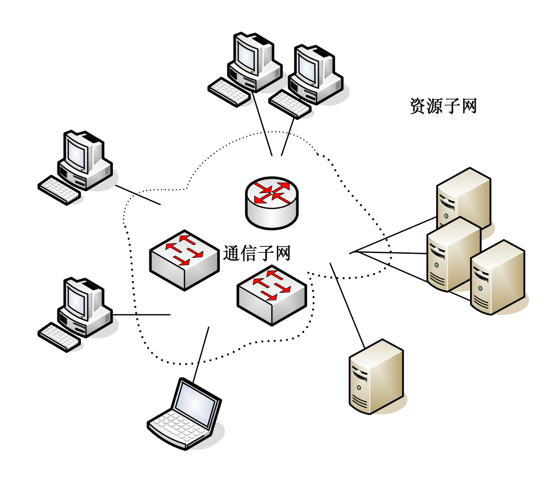


图 10-3　计算机网络组成示意图

1．服务器

服务器（Server）就是指专门为网络中其他计算机提供服务的计算机。根据用户的需要，可以架设提供文件共享、统一存储与管理的文件服务器；提供打印服务的打印服务器；安装了数据管理系统，实现数据库服务的数据库服务器；提供 WWW 服务的 Web 服务器；提供电子邮件服务的 E-mail 服务器等。

服务器通常是一台性能较高的机器，由网络操作系统、相应的服务软件构建而成。通常用来作为服务器的计算机，需要配置更快的 CPU、网卡，更大的内存、硬盘，而显卡、声卡等非关键设备则可以配置差些。因此，服务器的处理速度是整个网络效能的瓶颈点。

如果在计算机网络中存在专用的服务器，就是“基于服务器的网络”，而在“对等网”中各台计算机之间是互为对方的服务器。

2．工作站

工作站（Workstation）也称为客户机，由服务器进行管理和提供服务、连入网络的任何计算机都属于工作站，其性能一般低于服务器。个人计算机接入 Internet 后，在获取 Internet 服务的同时，其本身就成为一台 Internet 网上的工作站。网络工作站需要运行网络操作系统的客户端软件。

在对等网中，有时一台工作站为另一台工作站提供服务，成为了临时的服务器，而有时则反过来。

3．信号的马路——传输媒体

计算机通信的基础是各种传输媒体，信号通过传输媒体传到它的另一端。传输媒体可以分为有线、无线两大类。

- 有线：双绞线、细/粗同轴电缆、光纤等。
- 无线：微波、红外、激光、卫星通信等。

传输媒体的选用直接影响到计算机网络的性质，而且直接关系到网络的性能、成本、架设网络的难易程度，下面针对主要的传输媒体做一简要的概述。

1）同轴电缆

同轴电缆是一种历史悠久的传输介质，在双绞线还未盛行之前，它几乎是计算机网络传输介质的霸主，广泛应用于各种计算机网络环境中。

同轴电缆有许多不同的规格，最常用的有细同轴电缆（Thin）RG 58 和粗同轴电缆（Thick）RG 11。细同轴电缆主要用于建筑物内的网络连接，而粗同轴电缆则常用于建筑物间的相连。它们的区别在于粗同轴电缆屏蔽更好，能传输更远的距离。

①细同轴电缆 RG 58

其最大传输距离为 185m，阻抗为 50Ω。其特点是电缆较细、弹性好、容易安装，而且连接方式非常简单，但它的传输距离比较短，超过 185m 后信号就会开始衰减，必须使用一些专用的设备（如中继器）来增强信号，但它的线材及连接成本均相当便宜，因此常用于室内的小型局域网架设。

注意：传输距离大于 185m 时，在 185m 处加上一个中继器，然后再连接新的一段电缆，但由于电气参数的限制，并不能够在一个网络中无限制地使用中继器。

- 整个网络中最多只能用4个中继器连接5个区域。
- 在这5个区域中，仅有1、2、5三个区域能连接计算机工作站。
- 另外两个区域仅用于延长传输距离，以增加网络总长度。

这就是著名的 5-4-3 规则。

②粗同轴电缆 RG 11

其最大传输距离为 500m，阻抗也是 50Ω。其特点是电缆较粗，因此弹性较差，而且制作方式较为复杂，在室内安装时会遇到麻烦；但它的最大传输距离远远大于 RG 58，可以达到 500m，所以常用于主干或建筑间连接。但要说明的是，由于现在网络技术的不断进步，并且这种电缆仅能提供 10Mb/s 的速度，所以主干或建筑间的连接逐渐被速度更快的光纤所替代。

2）双绞线

相对而言，双绞线的广泛应用比同轴电缆要晚得多，但由于它提供了更高的性价比，所以深受广大用户的青睐，加上当今的许多网络技术都是基于此进行开发的，所以它就更加快速地走进了市场，成为现在应用最广泛的铜基传输媒介。

根据双绞线外是否多加一层外皮包覆，可将双绞线分为两大类，无屏蔽双绞线（UTP）和有屏蔽双绞线（STP），它们的最大传输距离都是 100m（注：现在有些新型的双绞线正在逐步提高它的最大传输距离限制）。由于价格低是双绞线的一个强有力的竞争特性，所以物美价廉的无屏蔽双绞线得到了市场的广泛认可。由于屏蔽双绞线则相对应用较少，所以在此主要介绍无屏蔽双绞线 UTP。

双绞线是由两条有绝缘外皮包覆的铜线相互缠绕在一起的，这两条对绞的线称为一个线对。这是双绞线最基本的度量单位。市场上广泛出现的一般是每条双绞线由 4 对绞线组成。美国电子工业协会与远端通信协会（EIA/TIA）制定了 UTP 电缆的“电缆等级”。它们主要的差别在于缠绕的绞距，通常两条线缠绕得越密，代表绞距越小，而传输较能也越好。

- 5类线：是一种向高速率发展的开始，最大传输速率为100Mb/s。
- 超5类：迎合千兆位网的出现而出现的新的线材。
- 6类线：新一代高速率线材，最大传输速率为1 000Mb/s。

3）光纤

光导纤维，是一种传输光束的细而柔韧的媒质，简称光纤。它是新近出现的一种新的传输媒介，由于它独特的性能，使其成为数据传输中最有成效的一种传输介质。在它出现的初期，由于价格居高不下，所以影响它的广泛应用。现今时代，人们对数据传输的速度要求越来越高，具有较高传输性能的光纤及连接设备正值大幅度的降价之际，所以其必将成为今后广泛应用的新一代传输媒介，取代双绞线在当今网络中的统治地位。

光纤用光脉冲代替电子信号来传输数据，它与电缆相比，具有频带更宽（常以 GB 为单位度量）、抗干扰性强、保密性强、传输速度快（轻松达到 1 000Mb/s）、传输距离长的特点。

光纤有单模和多模之分。单模光纤采用窄芯线，使用激光作为发光源，所以其耗散极小；另外，激光是以一个方向射入光纤的，而且仅有一束，使用其信号比较强，可以应用

于高速度、长距离的应用领域中，但也使得它的成本相对更高。而多模光纤则更广泛地应用于短距离或相对速度更低一些的领域中，它采用 LED 作为光源，使用宽芯线，所以其耗散较大；再加上整个光纤内有以多个角度射入的光，所以其信号不如单模光纤好，也正是这样，相对低廉的价格是它的优势。

4）有线传输媒介比较与选择

同轴电缆、双绞线与光纤相互，各有优劣，各有适应的环境。它们之间的异同与对比如表 10-1 所示。

表 10-1　有线传输媒体对比表

线缆名称	传输距离	传输速度	成　本	安　装	抗干扰性
细同轴电缆	185m	10Mb/s	最低	容易	较强
粗同轴电缆	500m	10Mb/s	较低	较难	强
屏蔽双绞线	100m	10Mb/s～1 000Mb/s	较低	容易	强
无屏蔽双绞线	100m	10Mb/s～1 000Mb/s	最低	最容易	最低
多模光纤	2km	51Mb/s～1 000Mb/s	次贵	最难	最强
单模光纤	2～10km	1～10Gb/s	最贵	最难	最强

一般来说，粗同轴电缆和屏蔽双绞线现在已经逐步淡出市场，在平时已经不再使用，所以建议大家不是在特殊的情况下，尽量不要使用它们。而其他的几种则各有所长。

- 单模光纤：它适用于对传输要求高的网络，或者作为网络主干或高速广域连接。
- 多模光纤：它适用于对传输要求比较高的网络，适合作为广域连接。
- 无屏蔽双绞线：最适合用于局域网布线，根据实际需要可选用5类或6类线。
- 细同轴电缆：它适用于以最低的价格建立一个最简单、最小型的工作组级小网络。

5）无线电波

除用于无线电广播、电视节目和移动通信外，无线电波还可以用于传输计算机的数据。使用无线电波网络经常被非正式地认为是运行在无线电频率上的，并且其传输也被称为 RF 传输。与使用线缆不同的是，使用这种 RF 传输的网络并不需要在传输双方拥有物理上的实际连接。作为替代，每台计算机带有一个天线，经过它来发送和接收信息。这个关键的天线可大可小，取决于所需的接收范围。例如，一个只在一幢大楼内的传输天线可以小到安装在计算机内。

无线电波传送并不沿地球表面弯曲，所以 RF 可以和卫星技术相结合，提供长距离的通信服务，当然这种形式的代价相当昂贵。

由于使用无线电波进行通信要占用一个专门的频率，所以使用它需要相关部门的批准才可以进行。另外，它被窃听的可能性很大，要在安全上另外花很大功夫。要注意的是，无线电波还会受到环境和天气的影响，而影响整体效果。

6）微波

超出无线电使用的频率范围的微波也能用于传输各种数据信号。虽然微波说到底也是无线电波的一种，但由于它们的工作性质完全不同，所以在此将它另列入专门的一类。

无线电波是向各个方向传播的，而微波则是集中于某个方向，这样可以有效地防止他人截取信号。并且微波还能用 RF 传送承载更多的信息。但是它不能透过金属结构，它在

传输时一般需要发送端与接收端之间无障碍存在。微波对环境与天气的影响相对不十分敏感，而且其保密性要比无线电波高得多。

7）红外线

红外线传输其实与我们并不陌生，各种电器使用的遥控器都基本上是使用红外线进行通信的。红外线一般局限在一个很小的区域内，并且经常要求发送器直接指向接收器。红外硬件与其他设备相对比较便宜，且不需要天线。

另外，大家一定能在许多新型主板上看到内置的红外线收发器。所以，在一些这样的情况下使用红外线进行通信，也是一种有用的选择。

8）激光

除此之外，一束光也能用于在空中传输数据。与微波通信系统极其类似，采用这种通信方式的两个站点都应拥有发送和接收装置。

激光发出的光束走的是直线，所以在发送方与接收方之间不能有障碍物，而且激光的光束并不能穿过植物、雨、雪、雾等，所以激光传送的局限性很大。

4．计算机的哨卡——网卡

网卡也称为网络适配器、网络接口卡（Network Interface Card，NIC），在局域网中用于将用户计算机与网络相连，大多数局域网采用以太网卡（Ethernet），如 NE 2000 网卡、PCMCIA 卡等。

网卡主要负责完成如下功能：

- 读入由其他网络设备传输过来的数据包，并将其变成计算机可以识别的数据，通过主板上的总线将数据传输到所需PC设备中（CPU、内存或硬盘）。
- 将PC设备发送的数据，打包后输送至其他的网络设备中。
- 代表着一个固定的地址（MAC地址）：网卡拥有一个全球唯一的地址，它是一个长度为48的二进制数，它为计算机提供了一个有效的地址（工作在数据链路层）。

5．勤快的“猫”——调制解调器 MODEM

调制解调器也称为 MODEM，俗称“猫”。它是一个通过电话拨号接入 Internet（或其他专用网络）的硬件设备。

由于计算机内部使用的是“数字信号”，而通过电话线路传输的信号是“模拟信号”，语言不通。因此，需要有一个翻译在中间搭桥牵线。而 MODEM 正是这个翻译，它的作用就是当计算机发送信息时，将计算机内部使用的数字信号转换成可以用电话线传输的模拟信号，通过电话线发送出去；接收信息时，把电话线上传来的模拟信号转换成数字信号传送给计算机，供其接收和处理。

调制解调器的速率一直随着技术的进步而提高，从最早的 2 400b/s 到 9 600b/s，发展到 14.4kb/s、28.8kb/s、33.6kb/s 最后达到 56kb/s。

为了使电话线上可以承载更快速的数据传输，电信运营商通过升级局端系统，发展出了 ISDN、ADSL 技术，相应也提供了对应的 ISDN 终端、ADSL MODEM 等设备，使得数据传输速度进一步提高。ISDN 可以达到 64Kb/s~128Kb/s，而 ADSL 下载速度更是能够达到 8Mb/s。

6．信号的加油站——中继器和集线器

计算机网络的信息是通过各种通信线缆传输的，但是在这一过程中，信号会受到干扰，产生衰减。如果信号衰减到一定的程度，信号将不能识别，计算机之间不能通信。那么如何解决这一问题呢？

- 使用中继器。它工作在物理层，当通信线缆达到一定的极限长度时，可以在中间连接一个中继器，将衰减了的信号放大后，再传送出去，以解决这一问题。不过现在通常采用综合布线系统，在网络规划时就避免这一情况的出现，因此中继器已很少使用。
- 使用集线器（HUB）。它其实就是一个多端口的中继器，工作在物理层。

7．网络间的关卡——网桥、路由器和网关

网桥（Bridge）也连接网络分支，但网桥多了一个“过滤帧”的功能，其工作在数据链路层。一个网络的物理连线距离虽然在规定范围内，但由于负荷很重，可以用网桥把一个网络分割成两个网络。这是因为网桥会检查帧的发送和目的地址，如果这两个地址都在网桥的这一半，那么该帧就不会发送到网桥的另一半，这就可以降低整个网的通信负荷，这个功能就叫“过滤帧”。

假如需要连接两种不同类型的局域网，那就得用路由器（Router），它可以连接遵守不同网络协议的网络。路由器能识别数据的目的地地址所在的网络，并能从多条路径中选择最佳的路径发送数据，路由器工作在网络层。

如果两个网络不仅网络协议不一样，而且硬件和数据结构都大相径庭，那么就得用工作在网络层之上的网关（Gateway）。不过，路由器与网关在一般的局域网中几乎派不上用场。

8．交换机

严格地说，“交换机”不是一种专业的说法。交换机这个概念是由商家“炒作”出来的。交换机的名称源于交换技术，它是一种针对集线器的不足应运而生的。要说明交换技术就需要说明集线器的工作原理，当集线器接收到从计算机发来的信号时，它对信号进行放大、重新定时，然后发向网络上所有的计算机，让目标计算机自己去判断、接收信号。显而易见的是，这样的做法，使得在整个线路上有许多是没有必要的信号，这样也就浪费了许多带宽。怎样避免这些带宽的浪费呢？这正是交换技术出现的原因。

具体来说，在一个小网络中，用“交换机”（这里指使用了交换技术的集线器）代替集线器，所有的计算机结点都与它连接。交换机记住整个小网络所有计算机结点的位置及如何到达这个结点。当信号发送到交换机的时候，交换机并不是简单地将信号放大、重新定时且向整个网络发送出去，它首先查看这个信号的目标结点，然后根据它的记录直接将这个信号发给目标结点，而不是向整个网络广播。很明显这样做大大提高了网络的利用率，而且还可以多个结点同时通信，所以大大提高了网络的速度。根据交换机工作的原理可以分为如下几种。

- 第二层交换机：工作在数据链路层，用来代替集线器的一种运用在小型网络中的设备。
- 第三层交换机：工作在网络层，它可以完成普通路由器的部分或全部功能。

- 高层交换机：工作在网络层之上，它可以在完成普通路由器的功能的基础上，实现一些特殊的功能。

9．网络中的游戏规则——标准与协议

通信协议是计算机网络的灵魂。它是为了使网络中的不同设备间能够实现数据通信，而预先制定的一整套通信双方相互了解和共同遵守的格式。

在计算机网络发展史上出现过的通信协议很多，现在还在广泛运用的主流通信协议如下。

- 概念框架：OSI网络分层协议。
- 事实标准：包括Internet的基石TCP/IP协议族、Windows局域网标准NetBIOS、Netware网的标准SPX/IPX协议族。

10.2　网络协议与标准

在计算机网络中有许多不同厂商提供的计算机设备、网络设备，它们是靠什么如此有序地完成通信任务的呢？要想成功地通信，就必须具有相同的语言。交流什么、怎样交流、何时交流，都必须有一个两方都能够互相接受的规则。这些规则的集合就称为协议。它可以定义两个实体间控制数据交换的规则集合。

简单地说，网络通信协议，就是计算机网络通信实体之间的语言，就像人与人之间通信、交流所使用的语言一样。类似的，不同的网络结构可能使用不同的网络协议。

10.2.1　OSI 网络层次参考模型

为了使得不同厂商提供的计算机设备、网络设备互联互通，国际标准化组织（International Standard Organization，ISO）在 1979 年建立了一个专门的分委员会来研究和制定一种开放的、公开的、标准化了的网络结构模型。这就是著名的“开放系统互联参考模型”（Open System Interconnection，OSI）的协议模型。它定义了一套用于连接异种计算机的标准框架。由于 ISO 组织的权威性，加上人们需要一个相互兼容、共同发展的，新的网络体系，所以 OSI 参考模型成为各大厂商努力遵循的标准。

时值今天，虽然许多网络协议并不完全与它一致，但却都是根据它来制定的，所以确保了它们的开放性和兼容性。从某种意义上说，OSI 参考模型已成为计算机网络协议的“金科玉律”。

1．OSI 模型特点

OSI 参考模型采用一种分层结构对网络中两点之间的通信过程进行理论化的描述。它并不规定支持每一层的硬件或软件的模型，但是网络通信的每个过程均能与某一层相对应。

标准的 OSI 参考模型把网络通信的结构分成七层（见表 10-2）：应用层（Application Layer）、表示层（Presentation Layer）、会话层（Session Layer）、传输层（Transport Layer）、网络层（Network Layer）、数据链路层（Data Link Layer）、物理层（Physical Layer）。

表 10-2　OSI 七层结构

7. 应用层（Application Layer）
6. 表示层（Presentation Layer）
5. 会话层（Session Layer）
4. 传输层（Transport Layer）
3．网络层（Network Layer）
2．数据链路层（Data Link Layer）
1. 物理层（Physical Layer）

提示：可以使用“All people seem to need data process”来记住七层，每个单词的第一个字母与每一层相对应。

除最低层物理层之外，每一层的功能都是建立在它的下层协议上的，每一层按照一定的接口形式向上一层提供一定的服务，而把实现这一服务的细节屏蔽。这样就可以保证每一层的工作与其他各层不重复，层次分明，既易于理解分析，又易于生产商提供相应的设备，这样每一层各司其职，经过逐层工作后，数据即可在网络上传输。OSI 只是一个通信框架，并不在具体的通信过程中起作用，真正的通信是由适当的软、硬件实现的，它定义了：

- 网络设备之间如何交互，如果使用不同的通信协议，如何通信。
- 网络设备决定何时发送数据的具体方法。
- 保证网络传输被正确接收的机制。
- 网络拓扑结构设计的依据。
- 如何确保网络设备提供一定的速率。
- 网络传输介质上数据流的含义。

2．物理层

物理层（见图 10-4）的所有协议就是人为规定了不同种类的传输设备、传输媒介如何将数字信号从一端传送到另一端，而不管传送的是什么数据。

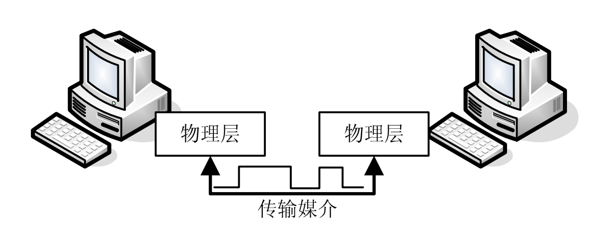


图 10-4　物理层原理示意图

它是完全面向硬件的，它通过一系列协议定义了通信设备机械的、电气的、功能的、规程的特征。

- 机械特征：规定线缆与网络接口卡的连接头的形状、几何尺寸、引脚线数、引线排列方式、锁定装置等一系列外形特征。
- 电气特征：规定在传输过程中多少伏特的电压代表“1”，多少伏特代表“0”。
- 功能特征：规定连接双方每个连接线的作用，用于传输数据的数据线，用于传输控制信息的控制线，用于协调通信的定时线，用于接地的地线。

• 过程特征：具体规定通信双方的通信步骤。

3．数据链路层

数据链路层（见图 10-5），在物理层已能将信号发送到通信链路中的基础上，负责建立一条可靠的数据传输通道，完成相邻结点之间有效地传送数据的任务。

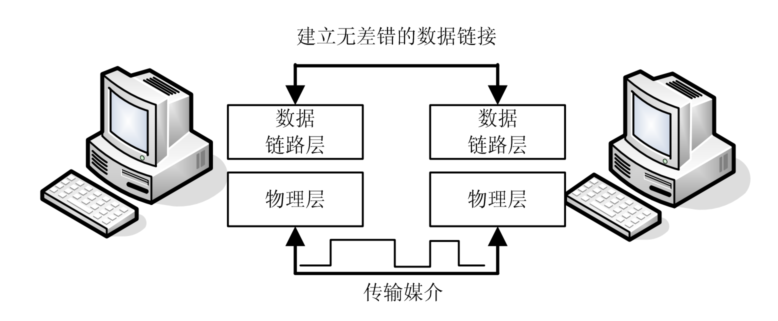


图 10-5　数据链接层原理示意图

正在通信的两个站在某一特定时刻，一个发送数据，一个接收数据。数据链路层通过一系列协议将实现如下功能。

• 封装成帧：把数据组成一定大小的数据块，我们称之为帧。然后以帧为单位发送、接收、校验数据。
• 流量控制：对发送数据的一方，根据接收站的接收情况，实时地进行传输速率控制，以免出现发送数据过快，接收方来不及处理而丢失数据的情况。
• 差错控制：对接收数据的一方，当接收到数据帧后对其进行检验，如果发现错误，则通知发送方重传。
• 传输管理：在发送端与接收端通过某种特定形式的对话来建立、维护和终止一批数据的传输过程，以此对数据链路进行管理。

就发送端而言，数据链路层将来自上层的数据按一定规则将比特流送到物理层处理；就接收端而言，它通过数据链路层将来自物理层的比特流合并成完整的数据帧供上层使用。最典型的数据链路层协议是 IEEE 开发的 802 系列规范，在该系列规范中将数据链路层分成了两个子层：逻辑链路控制层（LLC）和介质访问控制层（MAC）。

• LLC层：负责建立和维护两台通信设备之间的逻辑通信链路。
• MAC层：就像交通指挥中心控制汽车通行的车道一样，控制多个信息复用一个物理介质。MAC层提供对网卡的共享访问与网卡的直接通信。网卡在出厂前会被分配唯一的由12位十六进制数表示的MAC地址，MAC地址可提供给LLC层来建立同一个局域网中两台设备之间的逻辑链路。

IEEE 802 规范目前主要包括如下内容。

• 802.1：802协议概论。
• 802.2：逻辑链路控制层（LLC）协议。
• 802.3：以太网的CSMA/CD（载波监听多路访问/冲突检测）协议。
• 802.4：令牌总线（Token Bus）协议。
• 802.5：令牌环（Token Ring）协议。

- 802.6：城域网（MAN）协议。
- 802.7：宽带技术协议。
- 802.8：光纤技术协议。
- 802.9：局域网上的语音/数据集成规范。
- 802.10：局域网安全互操作标准。
- 802.11：无线局域网（WLAN）标准协议。

4．网络层

网络层，用于从发送端向接收端传送分组，负责确保信息到达预定的目标。看到这里，也许读者会觉得不可思议，数据链路层不是已经保证了相邻结点之间无差错传送数据帧了吗？那么网络层到底有什么用呢？其实，它存在的主要目的就是解决如下问题。

- 通信双方并不相邻。在计算机网络中，通信双方可能是相互邻接的，但也可能并不是邻接的，这样在一个数据分组从发送端发送到接收端的过程中，就可能在这个中间要经过多个其他网络结点，这些结点暂时存储"路过"的数据分组，再根据网络的"交通状况"选择下一个结点将数据分组发出去，直到发送到接收方为止。
- 正如前面所阐述的一样，由于OSI参考模型是出现在许多网络协议之后的，它就必须为使用这些已经存在的网络协议的计算机网络之间的相互通信做出贡献。事实上，网络层的一些协议解决了这样的异构网络的互联问题。

工作在网络层上的协议主要有 IP 协议和 IPX 协议，其工作原理如图 10-6 所示。

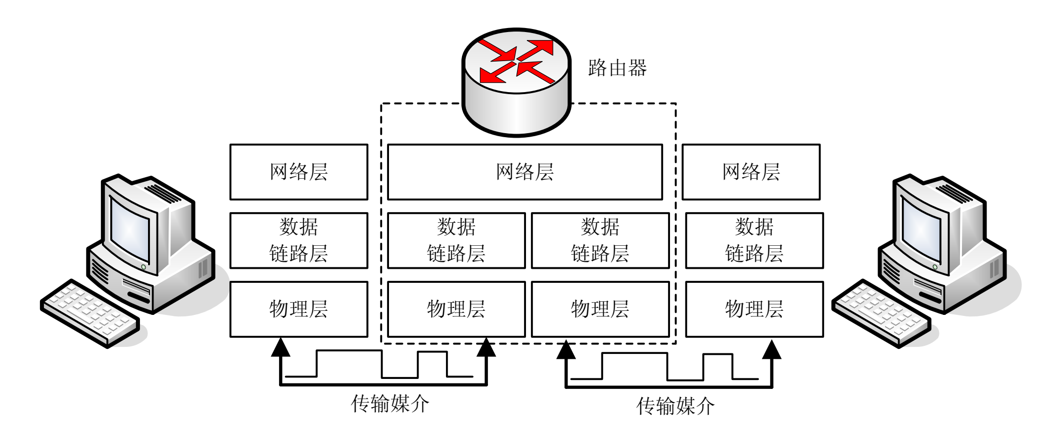


图 10-6　网络层原理示意图

5．传输层

传输层，实现发送端和接收端的端口到端口的数据分组传送，负责保证实现数据包无差错、按顺序、无丢失和无冗余地传输。在传输层上，所执行的任务包括检错和纠错。它的出现是为了更加有效地利用网络层所提供的服务。它主要体现在如下两方面。

- 将一个较长的数据分成几个小数据报发送。实际在网络上传递的每个数据帧都有一定大小限制。假设要传送一个字串"123456789"，但是它太长了，网络服务程序一次只能传送一个数字（当然在实际中不可能这么小，这里仅是为了方便讲解做的假设），因此，网络就需要将其分成9次来传递。就发送端而言，当然是从1传到9，但是由于每个数据分组传输的路径不会完全相同（因为它要根据当时的网络"交通状

况”而选择路径），先传送出去的包，不一定会先被收到，因此接收端所收到的数据的排列顺序与发送的顺序不同。而传输层的协议就给每一个数据组加入排列组合的记号，以便接收端能根据这些记号将它们“重组”成原来的顺序。

- 解决通信双方不只有一个数据连接的问题。这个问题从字面上可能不容易理解，实际上就是指，比如用电脑与另一台电脑连接拷贝数据的同时，又通过一些交谈程序进行对话。此时，拷贝的数据与对话的内容是同时到达的，传输的协议还负责将它们分开，分别传给相应的程序端口，这也就是端到端的通信。

工作在传输层的协议有：TCP、UDP、SPX，其中 TCP 和 UDP 都属于 TCP/IP 协议族（关于 TCP/IP 协议族在后面章节将会详细叙述）。

6．会话层

会话层主要负责管理远程用户或进程间的通信。该层提供如名字查找和安全验证等服务，允许两个程序能够相互识别并建立和维护通信连接。会话层还提供数据同步和检查点功能，这样当网络失效时，会对失效后的数据进行重发。在 OSI 参考模型中，会话层的规范具体包括如下内容。

- 通信控制。
- 检查点设置。
- 重建中断的传输链路。
- 名字查找和安全验证服务。

7．表示层

表示层以下的各层只关心从源地到目的地可靠地传输数据，而表示层则关心的是所传送信息的语义与语法。它负责将收到的数据转换为计算机内的表示方法或特定的程序的表示方法。也就是说，它负责通信协议的转换、数据的翻译、数据的加密、字符的转换等工作。在 OSI 参考模型中表示层的规范具体包括如下内容。

- 数据编码方式的约定。
- 本地句法的转换。

各种表示数据的格式的协议也属于表示层，如 MPEG、JPEG 等。

8．应用层

应用层就是直接提供服务给使用者的应用软件的层，比如电子邮件、在线交谈程序都属于应用层的范畴。应用层可实现网络中一台计算机上的应用程序与另一台计算机上的应用程序之间的通信，而且就像在同一台计算机上一样。在 OSI 参考模型中应用层的规范具体包括如下内容。

- 各类应用过程的接口。
- 提供用户接口。

9．OSI 参考模型的工作模式

首先，发送端由应用层的软件产生通信数据，然后各个层均对这些数据进行相应的处理，最后将它转换成比特流，通过物理上的传输介质来传送到接收端。接收端从物理层获得比特流，然后逐层分析，最后发给相应程序的数据，传给相应程序。在这个过程中，数据有很大的变化，具体如图 10-7 所示。

图 10-7　用 OSI 参考模型来传送数据的过程

10．OSI 参考模型小结

表 10-3 所示为 OSI 参考模型小结。

表 10-3　OSI 参考模型小结

层	功 能 描 述	对 应 协 议
应用层	用户接口，具体的网络应用	HTTP、Telnet、FTP、SMTP、NFS……
表示层	主要是定义数据格式，加密也属于该层	JPEG、ASCII、GIF、DES、MPEG……
会话层	定义了如何开始、控制和结束一个会谈，包括对多个双向消息的控制和管理，以便在只完成连续消息的一部分时可以通知应用，从而使得表示层看到的数据是连续的	RPC、SQL、NFS……
传输层	包括是否选择差错恢复协议，还是无差错恢复协议，这一层还在同一主机上对不同应用的数据流输入进行复用，还完成数据包的重新排序功能	TCP、UDP、SPX……

续表

层	功 能 描 述	对 应 协 议
网络层	该层对端到端的包进行定义。为了实现端到端的包传输功能，网络层定义了能够标识所有端点的逻辑地址。为了包能够正确地传输，还定义了路由实现方式和路由学习方法，同时还定义了包的分段方法	IP、IPX
数据链路层	该层定义了在一个特定的链路或媒体上获取数据	IEEE 802.3/.2、HDLC、PPP、ATM……
物理层	定义了有关传输媒体的物理特性的标准	RS232、V.35、RJ -45、FDDI……

10.2.2　局域网协议

局域网技术由于具有规模小、组网灵活和结构规整的特点，所以极易形成标准。事实上，局域网技术也是在所有计算机网络技术中标准化程序最高的一部分。国际电子电气工程师协议 IEEE 早在 20 世纪 70 年代就制定了三个局域网标准：IEEE 802.3（以太网，CSMA/CD）、802.4（令牌总线，Token Bus）、802.5（令牌环，Token Ring）。由于它已被市场广泛接受，所以 IEEE 802 系列标准已被 ISO 采纳为国际标准。而且随着网络技术的发展，又出现了如 802.7（FDDI）、802.3u（快速以太网）、802.11（无线局域网）、802.12（100VG-AnyLAN）、802.3z（千兆以太网）等新一代网络标准。局域网协议是工作在数据链路层上的。

10.2.3　广域网协议

在地域分布很远、很分散，以致无法用直接连接来接入局域网的场合下，广域网（WAN）通过专用的或交换式的连接把计算机连接起来。这种广域连接可以通过公众网建立，也可以通过服务于某个专门部门的专用网建立起来。

相对来说，广域网显得比较错综复杂，主要是用于广域传输的协议比较多：PPP（点对点协议）、DDN、ISDN（综合业务数字网）、X.25、FR（帧中继）、ATM（异步传输模式）等。下面就逐一简要地叙述，以便读者更好地了解和选择广域网协议。

1．PPP 点对点协议

PPP 点对点协议主要用于“拨号上网”这种广域连接模式。一般来说，一些无法使用专门的网络线连接的双方（比如家庭用户、移动用户）需要广域相连接时，就可以借助分布最广的公用交换电话网来实现，如图 10-8 所示。

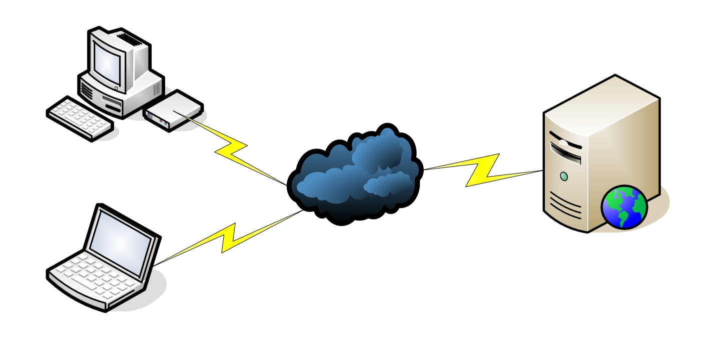


图 10-8　“拨号上网”示意图

如图 10-8 所示，终端通过调制解调器的调制，将要传输的数字信号调制成模拟信号，然后通过模拟的 PSTN 线路传输到目的地。

平时用户要浏览互联网上的网页时，首先通过调制解调器连接到电话线上，然后将在远方服务器的内容通过电话线传送到自己的计算机中来。或者当大家要发送电子邮件时，就将写好的电子邮件从电话线中传送出去。

另外，两个不同城市的两台计算机要互相传送数据，也可以通过装在两台计算机上的调制解调器，让其中一台呼叫另一台（拨打它的电话号码），而建立点对点的连接来实现的。

迄今为止，拨号上网还是绝大多数的家庭用户和小型办公室用户广域连接的一种最常用的手段。但是因为传输线路是模拟线路，所以传输速度较慢。

2．ISDN 综合业务数字网

ISDN 经历了一个极为漫长的“进化”过程。在它出现的时候，远程通信界的专家们都声称它是未来的公共电话、电信接口。但是它的不够经济却严重地阻碍了其得到广泛应用。

中国电信用了一个形象的名称——“一线通”描述出了它的特点：ISDN 将数据、声音、视频信号集成进一根数字电话线路，提供了有效经济的途径，将用户与高带宽数字服务相连。

ISDN 可分为 N-ISDN（窄带 ISDN）和 B-ISDN（宽带 ISDN）两种。

其中常用于家庭及小型办公室的是 N-ISDN，它提供的基本速率接口（BRI）服务由两个 B 信道和 1 个 D 信道组成（2B+D），其中 B 信道为 64Kb/s，D 信道为 16Kb/s。

而 B-ISDN 提供的主要速率接口（PRI）则根据不同的国家而不尽相同。在北美、日本为 23 个速率为 64Kb/s 的 B 信道和 1 个速率也为 64Kb/s 的 D 信道，总速率为 1.544Mb/s，即 23B+D。而在欧洲、澳洲及其他国家，一般则由 30 个速率为 64Kb/s 的 B 信道和 1 个速率也为 64Kb/s 的 D 信道构成，总的接口速率可达到 2.048Mb/s，也就是 30B+D。

3．xDSL

xDSL 是 DSL（Digital Subscriber Line）的统称，意即数字用户线路，是以铜电话线为传输介质的传输技术组合。DSL 技术主要分为对称和非对称两大类。

- HDSL（高速对称DSL）：是xDSL技术中最成熟的，它利用两对双绞线传输，支持Nx64Kb/s和多种速率，最高可达E1速率。
- SDSL（对称DSL）：利用单对双绞线传输，支持多种速率，最高到T1/E1。
- MVL：Paradyne公司开发的低成本对称DSL传输技术，可以提供上下行768Kb/s，传输距离可达6km。
- ADSL（非对称DSL）：利用现有铜双绞线（即普通电话线），提高到24Mb/s下行速度（目前国内一般为8Mb/s），3.5Mb/s上行速度，传输距离为3～5km。

4．DDN 数字专线

我国邮电部于 1994 年 10 月完成了全国数字数据骨干网的一期建设。这个网络是利用光纤、数字微波或卫星数字交驻连接设备组成的数字数据业务网。这些数字线路用于出租给最终用户。

由于在用户使用 PPP 协议拨号上网时，发送、接收数据所通过的电话线路不明确，根据当时线路的拥塞情况不同而不同，所以它的传输是低速且不稳定的。

而某些用户需要更高的传输速度和质量，就可以租用 DDN 线路来实现。租用了 DDN 线路，就等于在用户与电信局端直接用一条定制带宽的专用电话线路相连，显然这能大大提高整个数据传输的稳定性和速度。这项业务开通后，受到了用户的广泛好评，并且广泛被采用。

在 DDN 的客户端需要一个称为 DDN MODEM 的 CSU/DSU 设备，以及一个路由器，它的价格与 DDN 线路的带宽相关。

5．X.25

X.25 是历史最悠久的广域数据传输协议。尽管它是所有广域数据传输协议的鼻祖，而且也曾经为广域传输做出了很大的贡献，然而现在它似乎已经走到了尽头，X.25 的应用越来越少见。

6．FR 帧中继

作为 X.25 网络协议的发展，帧中继是一种高性能的广域网协议。它是 X.25 的一个简化版本，它省去了 X.25 的一些强制功能，如提供窗口技术和数据重发功能，这是因为帧中继的设计是以网络的传输环境已经有了很大的提高为前提的。

1990 年，Cisco、Digital Equipment、Northern TeleCom 和 StartaCom 等公司组成一个联合体，共同开发了帧中继技术。此后，帧中继技术有了迅猛发展。

从整个连接上，帧中继与 X.25 相类似。但它在数据分组确认和差错校验方法有了很大的简化，而且分组的转发也有了改变。帧中继只要接到分组头，就开始转发，这样进一步提高了速度。但是，需要强调的是，帧中继在网络环境不好的情况下，将无法像 X.25 那样提供较好的传输质量，而且可能会使传输质量急剧下降。

7．ATM 异步传输模式

ATM 是这几年兴起的一种宽带网络技术。许多业界人士都认为 ATM 技术给计算机网络带来巨大的革新。甚至有些商家认为它是这 10 年来最有意义的网络技术。

虽然在这里将 ATM 技术划在广域网部分来介绍，但 ATM 却可以将局域网功能、广域网功能、语音、视频和数据集成进一个统一的协议。正是它的高度统一性和良好的可扩展性，给计算机网络技术掀开了新的一页，它具有如下优点。

- 速度：ATM支持高达622Mb/s的传输率。
- 可扩展性：ATM允许在现存结构中增加带宽和端口密度。
- 高传输质量QoS：它保证了传输服务的QoS，这也是一般网络技术所不具备的。
- 一体化安装：ATM提供了端到端解决方案的潜力，这意味着它的应用可以从桌面到局域网，一直延伸到广域网。

根据 ATM 技术的特点与其约束，它可以适合于如下几种应用。

- 由于ATM技术提供了基于专用带宽的设计和数据优先级设计，使得它特别适合多媒体和视频应用。
- ATM技术具有良好的扩展能力及高性能的网络传输能力，适合构架骨干网。
- ATM具有高性能的无缝集成广域网和局域网的能力，所以被广泛地应用于广域网建设中。

10.2.4　互联网协议

世界上最大的广域资源网就是 Internet，即互联网。它的通信协议基础就是著名的 TCP/IP 协议族。在后面的章节中将详细介绍。

10.3　网络结构与通信

计算机网络的结构又称为拓扑结构，通常包括三种基本形式：总线型拓扑、星型拓扑和环型拓扑。其他的拓扑形式都是从这三种拓扑结构中衍生而来的。

10.3.1　总线型拓扑结构

总线型拓扑结构，顾名思义，就是指在这种拓扑结构中所有的电脑用电缆将整个网络从头串到尾。这是所有的网络拓扑结构中最基本、也是最简单的一种，它的结构如图 10-9 所示。

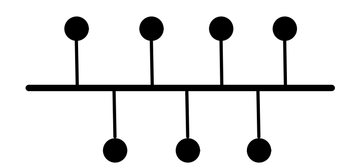


图 10-9　总线型拓扑结构示意图

这种拓扑结构具有所需电缆少、布线容易、单点可靠性高的优点，不过它也存在着一些不足。

- 故障诊断困难：由于在总线结构中，只要有一个结点失效，将引起整个网络失效。所以出现故障时，必须一个结点一个结点地检测，以便发现问题之所在。
- 对站点要求较高：因为接在总线上的所有站点发送和接收的数据都通过共用的总线，所以每个站点要有介质访问控制功能，以便与其他站点有序地共享总线。因此，增加了每个站点的软/硬件费用。

10.3.2　星型拓扑结构

星型拓扑结构，是由中央结点和通过点到点链路接到中央结点的各站点组成的，是现在用得最多的一种网络拓扑结构。它的结构如图 10-10 所示。

整个网络由中央结点执行集中式通信控制管理，因此中央结点相当复杂，而各个站点的通信处理的负担都很小。一般在星型拓扑结构的中央结点是一个称为集线器（或交换机）的设备，其负责将各个站点广播转发，或直接转发给接收方结点，这根据其复杂性不同而不同。

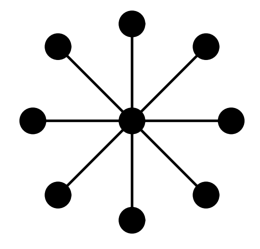


图 10-10　星型拓扑结构示意图

这种拓扑结构具有如下优点。

- 整体可靠性高：由于在星型拓扑结构中，每一个连接只连接一个设备，所以当连接出现故障时不会像总线型那样全线瘫痪，而只影响一个设备，这样就使整个网络具有较高的整体可靠性。
- 故障诊断容易：由于每个站点都直接连接到中央结点上，所以，故障十分容易检测和隔离。只要确定哪个站点通信出现问题，就能确定出故障的通信连接。
- 对站点要求不高：由于每一个站点都占用了一条专有的连接，所以不存在控制如何访问传输媒介的问题。这样就不像总线型网络那样需增加这方面的软件。

就像世界上任何事物一样，有利就有弊，星型拓扑结构虽然解决了不少问题，但也带来了新的不足。

- 所需电缆多：由于每个站点均需要专有的电缆与中央结点相连，所以整个网络需要使用更多的电缆。
- 整个网络可靠性依赖于中央结点：很明显，如果星型网络的中央结点出现故障，那么全网也就不可能工作。

10.3.3 环型拓扑结构

环型拓扑结构，顾名思义，就是指所有站点被绕成一圈的电缆所连接起来，整个结构看起来像一个圆圈，它的结构如图 10-11 所示。

整个网络的电缆绕成一圈，整条电缆并没有头尾之分。从串接的方式上看，与总线型拓扑结构相当类似，同样是由一条条电缆将相邻两个站点连接起来。但它的信号传递方式却大不相同。在环型拓扑结构中，环中有一个控制发送数据权力的“令牌”，它在环中流动。如果站点要发送数据，要先等待空的“令牌”到来，然后将要发送的数据附在“令牌”的后边，绕环传送，经过的每一个站点都接收、判断。如果是发给它的则接收，否则将数据再次送往环中的下一站，如此周而复始。

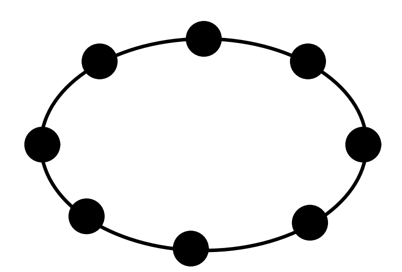


图 10-11　环型拓扑结构示意图

这种拓扑结构具有如下优点。

- 所需电缆较少：环型拓扑结构也是共享传输介质的，所以所需的电缆与总线型拓扑结构一样，比较节省电缆。
- 适用于光纤：环型拓扑结构是单方向传输数据的，这个特点与光纤“脾气相同”。

它存在着如下方面的不足。

- 整体可靠性差：由于所有的站点是一个挨着一个相连的，如果每一个结点之间的连接出了故障，则整个网络的通信也就中断了。

- 诊断故障困难：同样道理，当网络的通信中断要检测原因时，由于任何一个结点出现问题都可能导致整个网络中断，所以也要挨个站点检测。
- 对站点要求高：由于在数据传输中，“令牌”起到决定性作用，因而它所有的网络接入设备较复杂，也比其他的网络接入设备昂贵。

10.3.4 其他拓扑结构

前面介绍的三种拓扑结构是最基本、最常用的计算机网络拓扑结构。但是由于计算机网络的使用族群越来越多，这些基本的拓扑结构已无法满足使用者的需要，这样就衍生出了一些混合的拓扑结构。主要有星型总线拓扑、星型环拓扑。

10.3.5 拓扑结构的选择

在计算机网络的实际架设中都离不开几种常用的拓扑原型，了解它们，对于研究和规划网络系统将很有好处。

如果所规划的网络，是一个电脑数量不多，而电脑所在位置相当集中，甚至在一个办公室中，网络间的传输量不大，当然用户可以不花什么心思去考虑用什么拓扑结构，因为使用哪一种都不会带来太大的差别。但是，计算机网络一旦规划、建立完成，往往有一个较长的生命周期，如果不预留下成长空间，则将是一种失败的网络规划。

具体地说，因为每一种拓扑结构都有其优缺点，当选择了一种拓扑结构虽然享受到它带来的优点，却也不自觉地运用了它的缺点。如果事先没对它所带来的缺点有所考虑，就可能使整个网络性能大打折扣，应考虑的主要因素有如下几点。

- 总成本：不管选用什么样的传输介质，都需要安装，安装费用的高低和拓扑结构的选择有密切的关系。
- 灵活性：当加入或移出结点时，不同的拓扑结构所花去的代价是不同的，有的易于改变配置，有的则十分困难。
- 可靠性：不同的网络拓扑结构在不同的环境下，其可靠性能会有很大差别。这个因素十分重要。

10.4 Internet 和 Intranet 基础

本节将介绍 Internet 和 Intranet 基础知识。

10.4.1 Internet 网络协议

在 20 世纪 70 年代中期，美国高级国防研究项目署（DAPRA）为了建立一个适应战争的联通全国军部的大型网络 APRANET，就开创了这种异种网络互联的先河。为了完成这个网络的建设，DAPRA 投入了大量的人力、物力，最后在许多大学的参与下，制定了一系列的协议，并且高效地完成了网络互联的任务。这一系列的协议就是著名的 TCP/IP 协议。

TCP/IP 协议是当今世界上最流行的开放系统协议集。它正在支撑着 Internet（国际互联网）的正常运转。下面介绍 TCP/IP 协议集的组成，如表 10-4 所示。

表 10-4　TCP/IP 协议集与 OSI 各层的对应关系

应用层	FTP　TELNET
表示层	SMTP　HTTP
会话层	SNMP
传输层	TCP UDP
网络层	ICMP IP 路由选择协议
数据链路层	ARP　RARP
物理层	任意

下面我们就一起来看看，它们是如何协作而将各种异构的网络互联起来，提供一个统一的通信体系结构的。

1．IP 及相关协议

由于各种网络协议主要定义了物理层和数据链路层。要让这些在最底两层不同的网络能够形成一个统一的通信大网，则必须在更高的一层——网络层得到统一。

相对应的，IP 协议（Internet Protocol）就是运行在网络层上，为实现这样的功能而设计的。它为这个统一的大网规定了地址访问信息及一系列相关的信息，它是整个 TCP/IP 协议集的最核心协议之一。

1）IP 地址

为了让连接在整个大网上的主机能够相互通信，IP 协议给每一台主机分配一个唯一的地址，这个地址就叫 IP 地址。

IP 地址的长度为 32 位，它分为网络号和主机号两部分。网络号标识一个网络，一般网络号由互联网络信息中心（InterNIC）统一分配。主机号用来标识网络中的一个主机，它一般由网络中的管理员来具体分配。一个由 32 位二进制数构成的 IP 地址是难以阅读的。为了平时更好地记忆和使用，人们就将它分成 4 组，每组 8 位，然后每组都以十进制表示，并用小圆点分开。这种表示方法又称为“点分十进制表示法”。例如：

IP 地址：　　11001010011001010110100101000010

分成 4 组：　11001010 01100101 01101001 01000010

用十进制数表示：202　　101　　105　　66

用小点隔开：　202　.　101　.　105　.　66

这样就得到了用点分十进制表示的 IP 地址：202.101.105.66。

2）IP 地址的分类

IP 地址分成网络号和主机号两部分，网络号部分所占字长就直接决定了整个互联网可以为多少个网络分配 IP 地址；主机号部分所占字长也直接决定了所包含网络中最大的主机数。然而，由于整个互联网所包含的网络规模可能比较大，也可能比较小，设计者最后聪明地选择了一种灵活的方案：将 IP 地址划分成不同的类别，每一类具有不同的网络号位数和主机号位数。

如图 10-12 所示，IP 地址的前 4 位用来决定地址所属的类别。

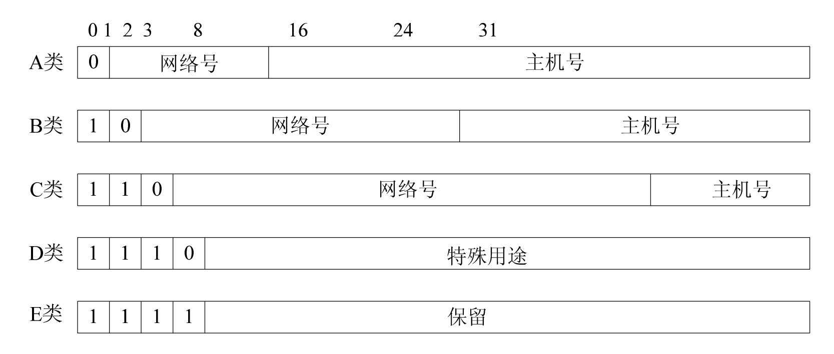


图 10-12　IP 地址分类

需要注意的是，在 IP 地址中，全 0 代表的是网络，全 1 代表的是广播。例如，假设一个单位的 IP 地址是 202.101.105.66，那么它所在的网络则由 202.101.105.0 来表示，而 202.101.105.255（8 位全为 1 转成十进制数为 255）则代表向整个网络广播的地址。另外，127.0.0.1 被保留作为本机回送地址。IP 地址类别对照表如表 10-5 所示。

表 10-5　IP 地址类别对照表

	A 类地址	B 类地址	C 类地址	D 类地址	E 类地址
地址格式	N.H.H.H	N.N.H.H	N.N.N.H	N/A	N/A
适用范围	大的组织	中型组织	小型组织	多目广播	保留
高位数字	0	10	110	1110	1111
地址范围	1.0.0.0 到 126.0.0.0	128.1.0.0 到 191.254.0.0	192.0.1.0 到 223.225.254.0	224.0.0.0 到 239.255.255.255	240.0.0.0 到 254.255.255.255
网络/主机位	7/24	14/16	21/8	N/A	N/A
最大主机数	16777214	65543	254	N/A	N/A

3）子网掩码

子网掩码是相对特别的 IP 地址而言的，如果脱离了 IP 地址就毫无意义。它的出现一般跟着一个特定的 IP 地址，用来为计算这个 IP 地址中的网络号部分和主机号部分提供依据，换句话说，就是在写一个 IP 地址后，再指明哪些是网络号部分，哪些是主机号部分。

子网掩码的格式与 IP 地址相同，对应的网络号部分用 1 填上，主机号部分用 0 填上。例如，一个 B 类地址：172.16.3.4，为了直观地告诉大家前 16 位是网络号，后 16 位是主机号，就可以附上子网掩码：255.255.0.0（11111111 11111111 00000000 00000000）。

①子网划分与 VLSM（可变长子网掩码）

随着网络的应用深入，IPv 4 采用的 32 位 IP 地址设计限制了地址空间的总容量，出现了 IP 地址紧缺的现象，而 IPv 6（采用 128 位 IP 地址设计）还不能够很快地进入应用，这时就需要我们采取一些措施来避免 IP 地址的浪费。以原先的 A、B、C 三类地址划分，经常出现 B 类太大、C 类太小或者 C 类都太大的应用场景，因此就出现了“子网划分”和“可变长子网掩码（VLSM）”两种技术。

子网划分就是利用主机号部分继续划分子网。子网可以用“子网掩码”来识别。例如，可以将一个 C 类地址划分子网，如图 10-13 所示。

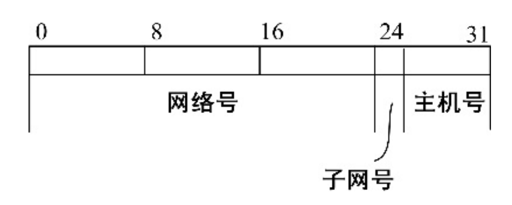


图 10-13　子网联网示意图

也就是将最后 8 位——原来的主机号，拿出 2 位用来表示子网，则可以产生 4 个子网，每个子网可包含 62 个主机（000001~111110，同样的 000000 代表网络，111111 代表广播被保留）。值得一提的是，此时，子网掩码就发生了变化：不是 255.255.255.0（11111111 11111111 11111111 00000000），而是 255.255.255.192（11111111 11111111 11111111 11000000）。

从 C 类地址中划分子网时就可以参照表 10-6 来进行。

表 10-6　C 类地址中的子网划分

主机号中用于表示子网号的位数	子网划分后相对应的子网掩码	总共可用的子网地址数	每个子网可用的主机地址数
2 位	255.255.255.192	4	62
3 位	255.255.255.224	8	30
4 位	255.255.255.240	16	14
5 位	255.255.255.248	32	6
6 位	255.255.255.252	64	2

采用了子网划分技术之后，虽然在一定程度上缓解了地址空间总容量受限这个问题，但又引发了一个新问题：即使得每个子网的主机数相等也难以有效地满足实际的需要，而且还引起了新的 IP 地址的浪费。VLSM 技术正是针对这个问题的行之有效的解决方案。

VLSM 是一种产生不同大小的子网的网络分配机制（在 RFC1878 中有详细说明）。VLSM 用直观的方法在 IP 地址后面加上“/网络及子网编码比特数”来表示。例如，192.168.123.0/26 就表示前 26 位表示网络号和子网号，即子网掩码为 26 位长，主机号为 6 位长。利用 VLSM 技术，可以多次划分子网，即分完子网后，继续根据需要划分子网。

为了帮助大家在考试时能够更快、更准确地计算出网络号/子网号、广播地址、可分配的网络/子网地址、有效子网号、主机数、子网数，下面对常见问题的解答技巧做一个总结。

- 基本子网划分，取网络号。A类保留第一个位，后面全0（如IP地址：10.1.0.0，网络号：10.0.0.0）；B类保留前两位，后面全0（如IP地址：131.2.3.0，网络号：131.2.0.0）；C类保留前三位，后面全0（如IP地址：192.168.1.5，网络号：192.168.1.0）。
- 复杂子网划分，取网络号。首先将掩码为255的部分对应的部分照抄，然后对非255部分，将掩码和IP地址均转成二进制进行“与”运算。例如，IP地址为192.168.1.100，子网掩码为255.255.255.240，则前三个数都照抄，而最后一部分先转二进制后再做“与”运算（0110 0100 AND 1111 0000 = 0110 0000，即96），得到192.168.1.96。
- 给定IP地址和掩码，计算网络/子网广播地址时，可根据规则：“网络/子网号是网络/子网中的最小数字，广播地址是网络/子网中的最大数字值，网络中有效、可分配的地址则是介于网络/子网号和广播地址之间的IP地址”。

- ➢ 基本子网划分，取广播地址。掩码为255的部分照抄，为0的部分改为255，例如，IP地址是131.1.0.4，子网掩码为255.255.0.0，则广播地址为131.1.255.255。
- ➢ 复杂子网划分，取广播地址。对于255部分照抄，0部分转为255，对于其他部分则先用256减去该值得到x，然后找到与IP地址中对应数最接近的x的倍数y，再将y减1即可。例如，IP地址为131.4.101.129，子网掩码为255.255.252.0，则首先将255、0的部分处理完，得到131.4.____.255，然后用256−252=4，与101最接近的4的倍数是104，因此得到广播地址为131.4.103.255。

- 复杂子网划分，获取有效子网数。例如，IP地址是140.140.0.0，子网掩码是255.255.240.0，则先找到特别的掩码位240，转换成二进制数11110000，因此得知主机位为4，再用24为基数进行增长：140.140.0.0，140.140.16.0，140.140.32.0，140.140.48.0……140.140.248.0。

②路由汇聚

路由汇聚（Route Summary）是一种有效简化路由表信息的手段，它将多个子网合并成超网，这样就能够用一条记录来表示多个子网。其工作原理比较复杂，不过考试中的题目其实很简单，通常是要求你根据子网的地址选择出汇聚后生成的网络地址，而解答时只需要选择一个包含这些子网地址的超网地址即可。

例如，设有下面 4 条路由：172.18.129.0/24、172.18.130.0/24、172.18.132.0/24 和 172.18.133.0/24，如果进行路由汇聚，能覆盖这 4 条路由的地址是（1）。网络 122.21.136.0/24 和 122.21.143.0/24 经过路由汇聚，得到的网络地址是（2）。（正确答案：A、B）

（1）A．172.18.128.0/21　　B．172.18.128.0/22
　　C．172.18.130.0/22　　D．172.18.132.0/23

（2）A．122.21.136.0/22　　B．122.21.136.0/21
　　C．122.21.143.0/22　　D．122.21.128.0/24

这是一道典型的路由汇聚的题目，但要注意能覆盖的地址和汇聚生成的地址有一些区别，汇聚生成的地址也是能够覆盖的，但是最小覆盖的。要解答这类题目，还是应该从 IP 地址中的网络号部分来进行判断，如表 10-7 所示。

表 10-7　地址的覆盖判断

地　址　项	IP　地　址	前　24　位	分　　析
题目地址 1	172.18.129.0/24	10101100 00010010 10000001	基准
题目地址 2	172.18.130.0/24	10101100 00010010 10000010	基准
题目地址 3	172.18.132.0/24	10101100 00010010 10000100	基准
题目地址 4	172.18.133.0/24	10101100 00010010 10000101	基准
选项 A	172.18.128.0/21	**10101100 00010010 10000**000	相同
选项 B	172.18.128.0/22	**10101100 00010010 100000**00	不同
选项 C	172.18.130.0/22	**10101100 00010010 100000**10	不同
选项 D	172.18.132.0/23	**10101100 00010010 1000010**0	不同

所谓的覆盖，就是指其网络号部分是相同的。从表 10-7 中可以发现题目中给出的 4 个地址只有前 21 位是相同的，因此只有选项 A 的地址是能够覆盖的。

而路由汇聚的判断，首先也必须进行覆盖性判断，如果有多个可以覆盖的，那么就需要选择最小覆盖（即网络号最长的那个）。从表 10-8 中不难得出只有选项 B 可以覆盖。

从上面的例题中，可以发现这类题目的核心是体会 IP 地址是由“网络号”+“主机号”组成的，一切题目都万变不离其宗，将其转成二进制数的格式，就不难得出正确的答案。

表 10-8　地址的覆盖判断

地　址　项	IP　地　址	前　24　位	分　　析
题目地址 1	122.21.136.0/24	01111010 00010101 10001000	基准
题目地址 2	122.21.143.0/24	01111010 00010101 10001111	基准
选项 A	122.21.136.0/22	**01111010 00010101 100010**00	不能
选项 B	122.21.136.0/21	**01111010 00010101 10001**000	覆盖
选项 C	122.21.143.0/22	**01111010 00010101 100011**11	不能
选项 D	122.21.128.0/24	**01111010 00010101 10000000**	不能

4）IPv 6

现在的 IP 协议的版本号为 4，所以也称为 IPv 4。它已经有了 20 年漫长的历史，为计算机网络互联做出了巨大的贡献。然而，互联网以人们不可想象的速度在膨胀，IPv4 不论从地址空间上，还是协议的可用性上都无法满足互联网的新要求。这样一个新的 IP 协议开始孕育而生，这个新版本 IP 协议，早先被称为 IPng，现在一般被叫做 IPv 6。

IPv 6 的设计要点在于克服 IPv 4 的地址短缺，无法适应对时间敏感的通信等缺点。值得一提的是，IPv 6 将原来的 32 位地址扩展成为 128 位地址，彻底解决了地址缺乏的问题。然而，由于 IPv 4 的广泛使用，而且充当重要的角色，一下子升级成新的协议是不大现实的，加上现在也出现了许多在 IPv 4 上的改良技术，使用 IPv 4 也能够应付现在的大部分网络互联要求。当然，随着时间的推移，新一代的 IP 协议将取代现有的 IPv 4，为网络互联提供一个更稳定、更优秀的协议平台。

5）ARP 地址解析协议

IP 地址是人为指定的，它并没有与硬件在物理上一对一联系起来。那么，如何将 IP 地址与硬件联系起来呢？我们都知道，每一台 PC 或每一个终端都有一个硬件地址（根据网络类型的不同而不同），只要用一种规则将 IP 地址与硬件地址相对应起来，而在数据链路层的一些设备已经具备使用一个特定的硬件地址进行通信的能力，那么 IP 地址也就与每一个通信实体一对一联系起来了。

将一台计算机的 IP 地址映射成相对应的硬件地址的过程叫地址解析，相应的，这个解析过程的规范被称为地址解析协议（Address Resolution Protocol，ARP）。

6）ICMP 互联网控制消息协议

IP 协议是一种尽力传送的通信协议，也就意味着其中的数据报仍可能丢失、重复、延迟或乱序传递。所以 IP 协议需要一种尝试避免差错并在发生差错时报告的机制。

TCP/IP 协议系列中包含了一个专门用于发送差错报文的协议，这个协议就叫作 Internet 控制报文协议（Internet Control Message Protocol，ICMP），这一协议对一个完全标准的 IP 是不可或缺的。有趣的是，这两个协议是相互依赖的：IP 在需要发送一个差错报文时要使用 ICMP，而 ICMP 却也是利用 IP 来传送报文的。

2．传输控制协议 TCP

传输控制协议（Transmission Control Protocol, TCP）是整个 TCP/IP 协议族中最重要的一个协议。它实现了一个看起来不太可能的事情：它在 IP 协议提供的不可靠数据服务的基础上，为应用程序提供了一个可靠的数据传输服务。

TCP 协议是怎样实现可靠性的呢？这可是一个十分复杂的问题。但说到底，最重要的是 TCP 采用了一个叫重发的技术。具体来说，就是 TCP 发送数据时，发送方通过一种重发方案来赔偿包的丢失，而且通信双方都要参与。在 TCP 传输过程中，发送方启动一个定时器，然后将数据包发出，当接收方收到了这个信息就给发送方一个确认。而如果发送方在定时器到点之前没收到这个确认，就重新发送这个数据包。

传输控制协议（TCP）作为 TCP/IP 协议族中最主要的协议之一，它为应用程序直接提供了一个可靠的、可流控的、全双工的流传输服务。在请求 TCP 建立一个连接之后，一个应用程序能使用这一连接发送和接收数据。TCP 确保它们按序无错传递。最终，当两个应用结束使用一个连接时，它们请求终止连接。

除此之外，由于互联网在不断变化，因此 TCP 的重发超时必须具有适应性。在具体实现中，TCP 协议是使用了缓冲、流控、窗口和拥塞控制等一系列机制来实现。感兴趣的读者可以专门阅读关于这方面的文献。

3．用户数据报协议 UDP

与 TCP 协议相对应的是用户数据报协议（User Datagram Protocol，UDP）。UDP 是一个简单的协议,它并没有显著地增加IP层的功能和语义。这为应用程序提供了一个不可靠、无连接的分组传输服务。因此，UDP 传输协议的报文可能会出现丢失、重复、延迟，以及乱序的错误，使用 UDP 进行通信的程序就必须负责处理这些问题。换句话说，就是采用 UDP 传输协议其实也无法避免前面提到的不可思议的工作量增加。

既然 UDP 有这样的缺点，那么它为什么还有存在的必要呢？其实，TCP 协议虽然提供了一个可靠的数据传输服务，但它是以牺牲通信量来实现的。也就是说，为完成一个同样的任务，TCP 会需要更多的时间和通信量。这在网络不可靠的时候，牺牲一些时间换来可靠是值得的，但当网络十分可靠的情况下，TCP 又成为浪费带宽的“罪魁祸首”，这时 UDP 则以十分小的通信量浪费占据优势。

另外，在某些情况下，每个数据的传输可靠性并不十分重要，重要的却是整个网络的传输速度。例如，语音传输，如果其中的一个包丢失了，重发也没有用，因为这个语音数据已经是失效的，谁能想象一个你先听到一分钟后的话音，再听到一分钟前的话音的通信？

所以，UDP 的存在是顺应一些特定的数据传输需要的。

10.4.2　Internet 应用

TCP 与 IP 协议为计算机网络提供了一个端到端通信的能力。而计算机网络的价值并不在计算机网络的本身，而是构建在它上面的各种各样的应用系统。在 Internet 上常用的应用包括如下方面。

1．DNS 域名服务

在用 TCP/IP 协议族架设的网络中，每一个结点都有一个唯一的 IP 地址，用来作为它们唯一的标志。然而，如果让使用者来记住这些毫无记忆规律的 IP 地址将是不可想象的。

人们就需要一种有记忆规律的字符串来作为唯一标记结点的名字。

虽然符号名对于人来说极为方便，但在计算机上实现却不是那么方便。为了解决这个需求，一个域名服务系统 DNS 应运而生，它运行在 TCP 协议之上，负责将字符名—域名转换成实际相对应的 IP 地址。这样，它就在不改变底层协议的寻址方法的基础上为使用者提供了一个直接使用符号名来确定主机的平台。经过了十余年的发展完善，DNS 已经成为了一套成熟的机制，广泛地应用于 Internet，为成千上万的人服务。

在域名的组织上，每台计算机的域名由一系列用“.”隔开的字母或数据构成的段组成。一个域名可以由几个段组成，它们是怎样被赋值的呢？由 InterNIC（域名分配机构）规定最高域的选择方法，然后由逐层的组织自己确定剩下的部分，如表 10-9 所示。

表 10-9　域名组织结构

域　　名	应　用　于
com	商业组织
edu	教育结构
gov	政府组织
mil	军事组织
net	主要网络支持中心
org	上述以外的组织
arpa	临时 ARPA 域
int	国际组织
国家代码	国家

2．WWW 万维网服务

提到互联网的使用，人们就一定会联想到大名鼎鼎的万维网服务（World Wide Web，WWW）。它是一个大规模在线式的信息储藏所，用户可以通过一个被称为浏览器的交互式应用程序来查找它所要的信息。

从技术上说，WWW 是一个支持交互式访问的分布式超媒体系统。超媒体系统直接扩充了传统的超文本系统。在这两个系统中，信息作为一个文档集而存储起来，除基本的信息外，还包含指向其他的文档。Web 文档用超文本排版语言（HTML）来撰写。除文本外，文档还包括指定文档版面与格式的标签。在页面中可以包含图形、音频、视频等各种多媒体信息。

可以这么说，Web 服务已经成为一种最佳的信息发布媒体，许多著名的人士都认识到它的重要性，甚至可以认为，Web 服务是继报纸、广播、电视之后的新一代媒体。而且它以其独有的快捷有效、传播范围广的特征席卷全球。

在 WWW 中，依赖于标准化的统一资源定位器（Uniform Resource Locator，URL）地址来定位信息的内容。在进行页面访问时，通常采用超文本传送协议（Hypertext Transfer Protocol，HTTP），其服务端口就是 HTTP 服务端口。

3．E-mail 电子邮件服务

作为当今互联网中最大的应用，E-mail 服务最初是被设计为传统的办公室备忘录的简单扩展。像办公室备忘录一样，电子邮件信息由一个人创建，副本发送给其他人。也像办

公室备忘录一样，电子邮件既方便、又不比普通通信开销大。

功能强大、使用简单的E-mail服务受到了大家的好评，以致许多用户将发送电子邮件到远地网点或从远地网点接收到电子邮件作为他们认识计算机网络的第一步。

4．FTP文件传输服务

在网络出现以前，当人们需要在不同的计算机之间进行数据传输时，唯一可以借助的工具是，诸如磁带、磁盘之类的磁介质。在一台计算机中将数据写入磁介质，然后将磁介质人为地拿到另一台计算机上，再将其中的数据传送。如果是长距离的交换，还需要将这个磁介质通过邮寄等方式来传送。当人们使用网络来传输数据的时候，才觉得这种方法是多么低效。

现在在Internet上使用最广泛的文件传输协议（File Transfer Protocol，FTP）。FTP允许传输任意文件，并且允许文件具有所有权与访问权限（用户可以指定哪些人能访问用户的哪些文件，甚至不能访问）。还有一个很重要的功能，它允许用户在IBM PC与Macintosh之间进行文件传输，这是一件多么激动人心的事呀！

基于FTP协议，用户可以架设一台专门供人们上传或下载文件的FTP文件服务器，还可以根据这些文件的性质对不同用户进行授权：将一些自己认为可以公开的内容开放给一些匿名用户（任何人），将一些不可以公开的内容，根据自己的实际情况给具备用户名和密码的用户。

文件传输服务提供了将整个文件副本从一台计算机传送到另一台计算机的功能，它日益成为许多计算机用户应用程序交流的好方法。正是这个原因，FTP服务也成为一种应用极为广泛的服务。TCP/IP协议族中包括两种文件传输服务：FTP和TFTP。FTP功能更强，它支持面向命令的交互界面，从而允许用户列。另外，TFTP是使用UDP协议进行实际的数据传输，而FTP则使用TCP协议进行实际的数据传输。

5．Telnet远程登录服务

在TCP/IP协议族中还包括一个简单远程终端协议——Telnet。Telnet允许某个网点上的用户与另一个网点上的登录服务器（提供Telnet服务的服务器）建立TCP连接。Telnet将用户键盘上的输入直接传递到远地计算机，好像用户是在连远程机器的本地键盘上操作一样。Telnet也将远地机器的输出送回到用户屏幕上。这种服务称为“透明”服务，因为它给人的感觉好像用户键盘和显示器是直接连在远程机器上的一样。

Telnet服务广泛应用于远程维护中，它使得维护一台远地的机器并不一定要在机器的面前，而只要通过网络，用TELNET远程登录进行相应的维护工作，当然有时这也成为了网络安全中的一个缺口。

10.4.3 Intranet基础

Intranet是基于Internet TCP/IP协议，使用WWW工具，采用防止外界侵入的安全措施，为企业内部服务，并有连接Internet功能的企业内部网络。

不同的企业会根据自己不同的需要组建Intranet。从技术角度来看，通常Intranet由网络、电子邮件、内部Web网、邮件列表、新闻组、远程访问、FTP等服务构成。而从企业的经营角度来看，Intranet通常包括如下内容。

- 企业内部主页：例如，工具和资源，搜索工具，索引和内容，表，电话本，企业服务宗旨，最新消息等。

- 通信处理：实现企业内部的个人通信，包括企业快报、公告栏、新闻等。
- 支持处理：包括人事处理、财会处理等。
- 产品开发处理：通常研究开发和工程两部分。
- 运作处理：企业经营的核心部分，通常包括采购、电子数据交换EDI、库存、制造，以及专门的服务开发等。
- 市场和销售处理：对销售人员进行支持。
- 客户支持：通过Web将信息给客户，接受网上的意见与投诉等。

10.5 网络管理基础

随着计算机网络技术的应用不断普及，它已经成为人们日常生活中的一部分，而且许多重要的信息化管理、服务系统也依托于计算机网络运行。因此，网络运行的稳定性、可靠性就显得十分重要，网络管理也就成为一个十分重要的内容。

由于网络越来越复杂，而且经常会出现一些不同供应商的设备混杂存在于同一个网络中，因此为了能够更好地进行管理，就需要有统一的网络管理标准。

1．客户机与服务器，管理员与代理

在传输协议和互联网协议中，并没有定义网络管理的功能，网络管理员用于监控网络设备的协议是在应用层运行的。也就是说，当管理员需要对特定硬件设备操作时，网络管理软件遵循传统的客户机/服务器模式。管理员计算机上的应用程序作为客户，网络设备上的应用程序作为服务器，它们之间的通信由现有的传输协议来建立。

但为了避免概念混淆，在网络管理应用中，不使用客户机、服务器这样的名称。而把在管理员的计算机上运行的客户程序称为管理员（Manager），把网络设备（被监控对象）上的服务器应用程序称为代理（Agent）。

2．网络管理的功能

根据 OSI 网络管理标准的定义，网络管理包括配置管理、性能管理、故障管理、安全管理、计费管理 5 个基本功能。

- 配置管理：自动发现网络拓扑结构，构造和维护网络系统的配置，包括配置的自动生成和备份功能。
- 故障管理：过滤、合并网络事件，有效地发生和定位网络故障。
- 性能管理：采集、分析网络对象的性能数据，对线路质量进行分析。
- 安全管理：结合用户认证、访问控制、数据传输、存储的保密与完整性机制，以保障网络管理系统本身的安全。
- 计费管理：对互联设备按IP地址进行流量统计，以便按用户要求实施计费。

3．SNMP

简单网络管理协议（Simple Network Management Protocol）是最早提出的网络管理协议之一。它一经推出就得到了包括 IBM、HP、SUN 等大型公司在内的广泛的应用和支持。现在已经成为这个领域的事实标准。

SNMP 的前身是 1987 年发布的简单网关监控协议（SGMP），最初是为了提供一种最小网络管理功能临时开发的，它具有两个主要的优点。

- 与SNMP相关的管理信息结构（SMI）及管理信息库（MIB）非常简单，从而能够迅速、简便地实现。
- SNMP是建立在人们都十分熟悉的SGMP的基础上的，拥有较多的操作经验。

SNMP 经历了两次主要的版本升级，最新的版本是 SNMPv 4。现在的版本在原来的基础上有了大幅提高，功能得到了很多增强，安全性方面有了很大的改善。

第 11 章

软件的知识产权保护

知识产权也称为“智力成果权”、“智慧财产权”。它是人类通过创造性的智力劳动而获得的一项权利。根据我国《民法通则》的规定，知识产权是指民事权利主体（自然人、法人）基于创造性的智力成果。知识产权具有无形性、专有性、地域性和时间性四大特点。

计算机软件具有固定的表达形式，容易复制等特征，大多数国家将其列为版权法的保护范畴，也是知识产权保护中的一个重要方面，因此作为一个软件从业人员，一方面应该了解法规，带头维护知识产权；另一方面也应学会利用知识产权维护自身的合法利益。

我国十分重视知识产权的保护，出台了一系列的相关法律法规。其中主要包括《著作权法》、《计算机软件保护条件》、《专利法》、《商标法》和《反不正当竞争法》。下面就针对这些主要的法律法规进行详细的解读。

11.1 著作权法及实施条例

1990 年 9 月通过，1991 年 6 月 1 日正式实施的《中华人民共和国著作权法》是知识产权保护领域的最重要的法律基础。另外国家还颁发了《中华人民共和国著作权法实施条例》作为执行补充，该条例于 1991 年 5 月通过，2002 年 9 月修订。在这两部法律法规中，十分详细、明确地对著作权保护及具体实施做出大量明确的规定。

11.1.1 著作权法客体

著作权法及实施条件的客体是指受保护的作品。这里的作品，是指文学、艺术和自然科学、社会科学、工程技术领域内具有独创性并能以某种有形形式复制的智力成果。

1．作品类型

其中包括以下 9 种类型。

- 文字作品：包括小说、诗词、散文、论文等以文字形式表现的作品。
- 口述作品：是指即兴的演说、授课、法庭辩论等以口头语言形式表现的作品。
- 音乐、戏剧、曲艺、舞蹈、杂技作品。
- 美术、摄影作品。
- 电影、电视、录像作品。
- 工程设计、产品设计图纸及其说明。
- 地图、示意图等图形作品。

- 计算机软件。
- 法律、行政法规规定的其他作品。

2．职务作品

为完成单位工作任务所创作的作品，称为职务作品。如果该职务作品是利用单位的物质技术条件进行创作，并由单位承担责任的；或者有合同约定，其著作权属于单位。那么作者将仅享有署名权，其他著作权归单位享有。

其他职务作品，著作权仍由作者享有，单位有权在业务范围内优先使用。并且在两年内，未经单位同意，作者不能够许可其他人、单位使用该作品。

11.1.2　著作权法主体

著作权法及实施条例的主体是指著作权关系人，通常包括著作权人、受让者两种。

1．著作权人与受让者

- 著作权人，又称为原始著作权人：是根据创作的事实进行确定的，创作、开发者将依法取得著作权资格。
- 受让者，又称为后继著作权人：是指没有参与创作，通过著作权转移活动成为享有著作权的人。

2．著作权人的确定

著作权法在认定著作权人时，是根据创作的事实进行的，而创作就是指直接产生文学、艺术和科学作品的智力活动。而为他人创作进行组织、提供咨询意见、物质条件或者进行其他辅助工作，不属于创作的范围，不被确认为著作权人。

如果在创作的过程中，有多人参与，那么该作品的著作权将由合作的作者共同享有。合作的作品是可以分割使用的，作者对各自创作的部分可以单独享有著作权，但不能够在侵犯合作作品整体的著作权的情况下行使。

而如果遇到作者不明的情况，那么作品原件的所有人可以行使除署名权以外的著作权，直到作者身份明确。

另外值得注意的是，如果作品是委托创作的，著作权的归属应通过委托人和受托人之间的合同来确定。如果没有明确的约定，或者没有签订相关合同，则著作权仍属于受托人。

11.1.3　著作权

1．著作权定义

根据著作权法及实施条例规定，著作权人对作品享有五种权利。

- 发表权：即决定作品是否公之于众的权利。
- 署名权：即表明作者身份，在作品上署名的权利。
- 修改权：即修改或者授权他人修改作品的权利。
- 保护作品完整权：即保护作品不受歪曲、篡改的权利。
- 使用权、使用许可权和获取报酬权、转让权：即以复制、表演、播放、展览、发行、摄制电影、电视、录像或者改编、翻译、注释、编辑等方式使用作品的权利；以及许可他人以上述方式使用作品，并由此获得报酬的权利。

2．著作权保护期限

根据著作权法相关规定，著作权的保护是有一定期限的。

1）著作权属于公民

署名权、修改权、保护作品完整权的保护期没有任何限制，永远属于保护范围。而发表权、使用权和获得报酬权的保护期为作者终生及其死亡后的 50 年（第 50 年的 12 月 31 日）。作者死亡后，著作权依照继承法进行转移。

2）著作权属于单位

发表权、使用权和获得报酬权的保护期为 50 年（首次发表后的第 50 年的 12 月 31 日），若 50 年内未发表的，不予保护。但单位变更、终止后，其著作权由承受其权利义务的单位享有。

3．使用许可

当第三方需要使用时，需得到著作权人的使用许可，双方应签订相应的合同。合同中应包括许可使用作品的方式，是否专有使用，许可的范围与时间期限，报酬标准与方法，违约责任。在合同未明确许可的权力，需再次经著作权人许可。合同的有效期限不超过 10 年，期满时可以续签。

对于出版者、表演者、录音录像制作者、广播电台、电视台而言，在下列情况下使用作品，可以不经著作权人许可、不向其支付报酬。但应指名作者姓名、作品名称，不得侵犯其他著作权。

- 为个人学习、研究或者欣赏，使用他人已经发表的作品。
- 为介绍、评论某一个作品或者说明某一个问题，在作品中适当引用他人已经发表的作品。
- 为报道时间新闻，在报纸、期刊、广播、电视节目或者新闻纪录影片中引用已经发表的作品。
- 报纸、期刊、广播电台、电视台刊登或者播放其他报纸、期刊、广播电台、电视台已经发表的社论、评论员文章。
- 报纸、期刊、广播电台、电视台刊登或者播放在公众集会上发表的讲话，但作者声明不许刊登、播放的除外。
- 为学校课堂教学或者科学研究，翻译或者少量复制已经发表的作品，供教学或者科研人员使用，但不得出版发行。
- 国家机关为执行公务使用已经发表的作品。
- 图书馆、档案馆、纪念馆、博物馆、美术馆等为陈列或者保存版本的需要，复制本馆收藏的作品。
- 免费表演已经发表的作品。
- 对设置或者陈列在室外公共场所的艺术作品进行临摹、绘画、摄影、录像。
- 将已经发表的汉族文字作品翻译成少数民族文字在国内出版发行。
- 将已经发表的作品改成盲文出版。

11.2　计算机软件保护条例

1991 年 6 月通过，10 月 1 日正式实施《计算机软件保护条例》是我国计算机软件保护的法律依据。该条例最新版本是在 2001 年年底通过，2002 年 1 月 1 日正式实施的。

由于计算机软件也属于《中华人民共和国著作权法》保护的范围，因此在具体实施时，首先适用于《计算机软件保护条例》条文规定，若是在《计算机软件保护条例》中没有规定适用条文的情况下，才依据《著作权法》的原则和条文规定执行。

11.2.1　条例保护对象

《计算机软件保护条例》的客体是计算机软件，而在此计算机软件是指计算机程序及其相关文档。

根据条例规定，受保护的软件必须是由开发者独立开发的，并且已经固定在某种有形物体上（如光盘、硬盘、软盘）。

另外要注意的是，其对软件著作权的保护只是针对计算机软件和文档，并不包括开发软件所用的思想、处理过程、操作方法或数学概念等。并且著作权人还需在软件登记机构办理登记。

11.2.2　著作权人确定

1．合作开发

对于由两个以上开发者或组织合作开发的软件，著作权的归属根据合同约定确定。若无合同，共享著作权。

若合作开发的软件可以分割使用，那么开发者对自己开发的部分单独享有著作权，可以在不破坏整体著作权的基础上行使。

2．职务开发

开发者在单位或组织中任职期间，所开发的软件符合以下条件的，则软件著作权应归单位或组织所有：

- 针对本职工作中明确规定的开发目标所开发的软件。
- 开发出的软件属于从事本职工作活动的结果。
- 使用了单位或组织的资金、专用设备、未公开的信息等物质、技术条件，并由单位或组织承担责任的软件。

3．委托开发

如果是接受他人委托而进行开发的软件，其著作权的归属应由委托人与受托人签订书面合同约定；如果没有签订合同，或合同中未规定的，其著作权由受托人享有。

另外，由国家机关下达任务开发的软件，著作权的归属由项目任务书或合同规定，若未明确规定，其著作权应归任务接受方所有。

11.2.3　软件著作权

1．软件著作权定义

根据《计算机软件保护条例》规定，软件著作权人对其创作的软件产品，享有以下九种权利。

- 发表权：即决定软件是否公之于众的权利。
- 署名权：即表明开发者身份，在软件上署名的权利。
- 修改权：即对软件进行增补、删节，或者改变指令、语句顺序的权利。
- 复制权：即将软件制作一份或者多份的权利。
- 发行权：即以出售或者赠与方式向公众提供软件的原件或者复制件的权利。
- 出租权：即有偿许可他人临时使用软件的权利。
- 信息网络传播权：即以信息网络方式向公众提供软件。
- 翻译权：即将原软件从一种自然语言文字转换成另一种自然语言文字的权利。
- 使用许可权、获得报酬权、转让权。

2．软件著作权保护期限

软件著作权自软件开发完成之日起生效。

1）著作权属于公民

著作权的保护期为作者终生及其死亡后的50年（第50年的12月31日）。对于合作开发的，则以最后死亡的作者为准。值得注意的是，在1991实施的上一版条例中，保护期限是25年，而在最新的条例中，则已经改为了50年。在作者死亡后，将根据继承法转移除署名权之外的著作权。

2）著作权属于单位

著作权的保护期为50年（首次发表后的第50年的12月31日），若50年内未发表的，不予保护。但单位变更、终止后，其著作权由承受其权利义务的单位享有。

3．合法复制品所有人权利

当得到软件著作权人的许可，获得合法的计算机软件复制品后，则复制品的所有人享有以下权利：

- 根据使用的需求，将该计算机软件安装到设备中（电脑、PDA等信息设备）。
- 可以制作复制品的备份，以防止复制品损坏，但这些复制品不得通过任何方式转给其他人使用。
- 根据实际的应用环境，对其进行功能、性能等方面的修改。但未经软件著作权人许可，不得向任何第三方提供修改后的软件。

4．使用许可的特例

如果使用者只是为了学习、研究软件中包含的设计思想、原理而以安装、显示、存储软件等方式使用软件，可以不经软件著作权人许可，不向其支付报酬。

5．侵权责任

根据计算机软件保护条件，侵犯软件著作权的法律责任包括民事责任、刑事责任、行政责任3种。

- 民事责任：包括未经软件著作权人许可，发表或登记其软件；将他人软件作为自己的软件发表或者登记；未经合作者许可，将合作开发产品视作自己单独完成的软件发表或者登记；在他人软件上署名或者更改署名；未经许可，修改、翻译软件。通常应承担停止侵害、消除影响、赔礼道歉、赔偿损失等民事责任。

- 行政责任：在前面的基础上，同时对社会公共利益造成损害的，行政管理部门可以没收其违法所得，没收复制品，处以罚款。
- 刑事责任：如果在未经授权的情况下复制或部分复制软件，或向公众发行、出租、传播软件，将处以每件100元，或货值金额5倍以下的罚款。如果是故意破解软件、故意删除或改变软件权利电子信息，非法转让不属于自己的软件著作权，则可以处以5万元以下的罚款。

另外，需要注意的是，如果是因为可供选用的表达方式有限，而造成与原来存在的软件相似，则不构成对原有软件著作权的侵犯。

11.3 商标法及实施条例

《中华人民共和国商标法》，自最早于 1963 年通过的第一版以来，已经做了多次修订，现在执行的是 2001 年 10 月 27 日通过、正式实施的。

11.3.1 注册商标

1．什么是商标

任何能够将自然人、法人及组织的商品与他人的商品区别开的可视性标志，就是可以用于注册的商标。商标可以包括文字、图形、字母、数字、三维标志和颜色组合。商标必须报商标局核准注册。通常包括商品商标、服务商标、集体商标，以及证明商标。

除一些与国家、政府、国际组织相同、相似的，以及一些带有民族歧视、影响社会道德等性质的标志不能够作为商标注册外，县级以上行政区划的地名也不能够作为商标。

2．商标的使用期限

商标的使用，是指将商标用于商品、包装、容器、交易文书、广告宣传、展览，以及其他商业活动中。

注册商标的有效期是 10 年，从核准通过，正式注册之日起开始计算。在有效期满之后，可以续注册，但必须在期满前 6 个月提出申请。未在此期间提出申请的，则给予 6 个月的宽限期，在宽限期还未提出申请的，将注销其商标。

3．注册商标的申请

要申请商标注册，应当按照公布的商品和服务分类表按类申请。每一件商标注册申请应当向商标局提交《商标注册申请书》1 份、商标图样 5 份（清晰，长宽不大于 10cm，不小于 5cm）；指定颜色的，并应当提交着色图样 5 份、黑白稿 1 份。

如果商标是三维标志，应在申请书中声明，并提交能够确定三维形状的图样。商标为外文或者包括外文的，应说明其含义。

如果有多个申请人，在同一天申请注册相同或近似的商标，则申请人应该提交其申请注册前在先使用该商标的证据，先使用者获得商标注册。如果都没有使用证据，那么将通过协商解决，协商无效，则通过抽签决定。

11.3.2 注册商标的专用权保护

注册商标的专用权，是以核准注册的商标和核定使用的商品有限的。而若存在以下行为之一，就属于侵犯注册商标专用权。

- 未经商标注册人的许可，使用相同或近似商标。
- 销售侵犯商标专用权的商品（注：如果销售方不知道是侵权商品，并且能够证明自己是合法取得的，不承担相应责任）。
- 伪造他人注册商标，或销售这些伪造的注册商标。
- 未经商标注册人同意，更换其注册商标，并将更换商标的商品投入市场。

当出现侵犯注册商标的专用权时，双方当事人可以协商解决。如果无法协商解决，可以向人民法院起诉，或提请工商局处理。法院可以根据侵权行为的情节判处50万元以下的赔偿。

11.3.3　注册商标使用的管理

当合法地注册商标使用权后，就可以在商品、商品包装、说明书或者其他附着物上标明“注册商标”或者注册标记（包括©和®）。

若商标注册人死亡或者终止，自死亡或终止之日起 1 年期满，而没有继续办理转移手续，任何人都可以向商标局申请注销该注册商标。

11.4　专利法及实施细则

《中华人民共和国专利法》是我国对专利技术保护的法律基础，最早在 1998 年 3 月 12 日获得通过，颁布实施。后来历经 1992 年 9 月 4 日、2000 年 8 月 25 日两次修订，现行的就是 2000 年通过，2001 年 7 月 1 日正式实施的版本。

11.4.1　专利法的保护对象

专利法的客体是发明创造，也就是其保护的对象。

1．发明创造的定义

这里的发明创造是指发明、实用新型和外观设计。

- 发明：就是指对产品、方法或者其改进所出的新的技术方案。
- 实用新型：是指对产品的形状、构造及其组合，提出的适于实用的新的技术方案。
- 外观设计：对产品的形状、图案及其组合，以及色彩与形状、图案的结合所做出的富有美感并适于工业应用的新设计。

2．授予专利权的条件

要想申请专利权的发明和实用新型，应当具备新颖性、创造性和实用性等特点。

- 新颖性：也就是在申请专利之前没有同样的发明或实用新型在国内外出现过（不过如果是自己在政府主办或承认的展会上展出、在规定的学术会议或技术会议上发表、他人未经同意泄露等情况，并不丧失新颖性）。
- 创造性：是指同原有的技术相比，有突出的特点和显著的进步。
- 实用性：是指其能够被制造或者使用，并且有积极的效果。

而对于想申请专利权的外观设计，应保证在国内外发表的外观设计不相同、不相近似。

值得注意的是，对于科学发现、智力活动的规则和方法、疾病的诊断和治疗方法、动植物品种及用原子核变换方法获得的物质，不能够被授予专利权。

11.4.2　确定专利权人

根据专利法的规定，专利权归属于发明人或者设计人，这是指对发明创造做出创造性贡献的人。对于在发明创造过程中，只负责组织、提供方便、从事辅助工作的都不属于发明人或设计人。

1．职务发明

如果是执行单位任务，或者是利用本单位的物质技术条件所完成的发明创造，被视为职务发明创造，通常包括：

- 在本职工作中做出的发明创造。
- 在履行单位交付的本职工作之外的任务所做出的发明创造。
- 退职、退休或者调动工作后1年内做的，与其原来承担的任务相关的发明创造。

对于职务发明的专利申请被批准后，单位是专利权人。对于利用单位的物质技术条件进行发明创造的，发明人、设计人与单位之间可以签订合同，重新规定专利权的归属。

2．合作发明、设计

对于合作发明、设计的，其专利权应属共同所有，但可以根据合作方之间另行签订的合同来确定专利权的归属。

3．委托发明

若一个单位或者个人接受其他单位或个人的委托，所完成的发明创造，若没有签订合同规定专利权归属，则专利权归属发明、设计者。

4．其他

如果非职务发明，则单位无权压制个人进行专利权申请。对于多个相类似的专利申请，则专利权归属最先提交的申请人。

11.4.3　专利权

1．专权保护

未经专利权人许可，实施专利的，就属于侵犯专利权，专利权人可以起起诉，申请调解。

- 假冒他人专利，没收违法所得，并处于3倍以下，或5万元以下的罚款，情节严重的，依法追究刑事责任。
- 以非专利产品冒充专利产品，责令整改，并可处以5万元以下的罚款。
- 侵犯专利权的赔偿数额，参照该专利许可使用费的倍数合理确定。
- 专利诉讼的有效期是2年，以专利权人得知侵权行为之日起计算。

对于以下情况，不视为侵犯专利权：

- 对于专利权人制造、进口或者经专利权人许可而制造、进口的专利产品，或者依照专利方法直接获得的产品售出后，使用、许诺销售或者销售该产品。
- 在专利申请日前已经制造相同产品、使用相同方法或者已经做好制造、使用的必要准备，并且在原有范围内继续制造、使用。
- 临时通过中国的国外运输工具，在其自身需要使用了专利。
- 专为科学研究和实验而使用有关专利的。

2．专利权保护期限

我国现行《专利法》规定的发明专利权保护期限为 20 年，实用新型和外观设计专利权的期限为 10 年，均从申请日开始计算。在保护期内，专利权人应该按时缴纳年费。

在专利权保护期限内，如果专利权人没有按规定缴纳年费，或以书面声明放弃其专利权的，专利权可以在期满前终止。

另外，任何单位和个人都可以在授予专利之日起，请求专利复审，如果复审未通过，则将终止专利权。

3．专利实施的强制许可

对于具备实施条件的单位，可以以合理的条件请求发明或者实用新型专利权人许可实施其专利。

若国家出现紧急状态或者非常情况时，可以为了公共利益强制实施发明专利、实用新型专利的许可。

11.5 反不正当竞争法

为了保护市场的公平环境，制止不正当竞争行为，我国在 1992 年 9 月 2 日通过，1993 年 12 月 1 日正式实施《中华人民共和国反不正当竞争法》。

11.5.1 不正当竞争

1．什么是不正当竞争

不正当竞争是指经营者违反本法规定，损害其他经营者的合法权益，扰乱社会经济秩序的行为。

- 采用不正当的市场交易手段：采用例如假冒他人注册商标；擅自使用与知名商品相同或相近的名称、包装，混淆消费者；擅自使用他人的企业名称；在商品上伪造认证标志、名优标志、产地等信息，从而达到损害其他经营者的目的。
- 利用垄断的地位，来排挤其他经营者的公平竞争。
- 利用政府职权，限定商品购买，以及对商品实施地方保护主义。
- 利用财务或其他手段进行贿赂，以达到销售商品的目的。
- 利用广告或者其他方法，对商品的质量、成分、性能、用途、生产者、有效期、产地等进行误导性的虚假宣传。
- 以低于成本价进行销售，以排挤竞争对手。不过对于鲜活商品、有效期将至及积压产品的处理，以及季节性降价，国清债、转产、歇业等原因进行降价销售均不属于不正当竞争。
- 搭售违背购买者意愿的商品。
- 采用不正当的有奖销售。例如，谎称有奖，却是内定人员中奖；利用有奖销售推销质次价高产品；奖金超过5 000元的抽奖式有奖销售。
- 捏造、散布虚伪事实，损害对手商誉。
- 串通投标，排挤对手。

2．保护条例

采用不正当竞争对别的经营者造成损害的，应承担赔偿责任。如果无法计算损失的，则赔偿侵权期因侵权所得的利润。

- 对于假冒注册商标、姓名、认证、产地的不正当竞争行为，可以根据《商标法》进行处罚；仿冒知名商标的，则可以根据情节罚款违法所得的1万～3万元罚款，特别严重的追究刑事责任。
- 通过贿赂达到销售目的，根据情节处以1万～20万元罚款，严重的追究刑事责任。
- 利用独占地位进行经营，根据情节处以5万～20万元罚款；借此销售质次价高商品的，则没收违法所得，并罚款1万～3万元。
- 采用广告误导消费者，处以1万～20万元罚款。
- 采用不合法的有奖销售的，根据情节处以1万～10万元罚款。
- 串通投标者，根据情节处以1万～20万元罚款。

11.5.2　商业秘密

1．什么是商业秘密

商业秘密是指不为公众所知，具有经济利益，具有实用性，并且已经采取了保密措施的技术信息与经营信息。在《反不正当竞争法》中对商业秘密进行了保护，存在以下行为的，视为侵犯商业秘密：

- 以盗窃、利诱、胁迫等不正当手段获取别人的商业秘密。
- 披露、使用不正当手段获取的商业秘密。
- 违反有关保守商业秘密的要求约定，披露、使用其掌握的商业秘密。

2．保护条例

对于侵犯商业秘密的，将根据情节处以 1 万～20 万元罚款。

第 12 章

计算机专业英语

本章主要讲解计算机专业英语。

12.1 综述

英语能力是软件设计师的必备能力，因此，计算机英语是软件设计师考试的重要内容。考试大纲要求“具有工程师所要求的英语阅读水平，理解本领域的英语术语”。在软件设计师上午试题中，共 75 分，其中英语占 5 分（2007 年以前英语占 10 分）。

1．软件设计师英语考试与其他英语考试的比较

近几年的软件设计师英语考试主要有如下几方面的特点。

- 难度略高于大学英语四级，相当于研究生入学考试。
- 题材限于计算机文化读物，不如其他英语考试广泛。
- 题型只限于短文填空（完型填空），题型单一。

2．复习与应试要点

根据考试试题的特点，软件设计师英语复习要点如下。

- 找一本研究生入学考试（或四级）英语复习资料，复习相关的固定搭配、短语、语法知识，重点复习其中的完型填空，掌握完型填空的考点及要求。
- 注意多读计算机报刊、杂志的时文，在了解这个领域最新信息的同时积累语言知识，训练阅读能力。在复习时看一些计算机英语材料，对这一领域的表达方式和词汇进行热身。本章精选了一些英语材料，供大家复习参考。
- 用近几年的软件设计师英语考试试题进行模拟测试，本章收集了近几年的试题。

由于软件设计师英语考试题型只限于短文填空（完型填空），因此，考生可以在考前作一些专项练习，结合复习总结出一些解题的技巧。一般可采用三步法，其要点如下。

- 粗略地看一遍全文，了解全文的信息。
- 以了解的信息作为基础，对全文进行精读，并进行完型填空。
- 从全局的角度，对答卷进行检查。

12.2 计算机专业英语词汇及缩略语精选

说明：计算机领域内的很多词汇的形式尚无统一规定。为统一起见，这里列出的词汇尽量去掉了时态、分词形式、复数等。对于一词多义的，尽量列出其最主要的、常用的意

思，以及在计算机领域内特定的意思。但对于某些词汇的分词、复数等形式在计算机领域中表达特定意义的，则做了保留。

12.2.1 常见计算机词汇

Abstract	**抽象的**
Abstraction	抽象
Acceptance test	验收测试
Acceptor	接收器
Access control	访问控制
Activation	活跃期
Active object	主动对象
Activity diagram	活动图
Actor	参与者
Actuator	传动器
Adapter	适配器
Addressing	寻址
Agent	代理
Aggregation	聚合
Agile Methodologies	敏捷方法学
Algebra	代数学
Algorithm	算法
Allocation	分配
Alphabet	字母表
Alphabetize	按字母顺序
Amplify	放大
Animation	动画
Antenna	天线
Architecture	构架
Argument	引数
Aspect oriented	面向方面的
Assembler	汇编程序
Assertion	断言
Assessment	评估
Association rule	关联规则
Association	关联
Assumption	假设
Asymmetric key encryption	非对称密钥加密
Atomicity	原子性
Attack tree	攻击树
Attribute	特性
Authentication	认证
Automation	自动控制化
Backdoor	后门
Backup	备份
Barrier	隔离层、隔离物
Baseline	基线
Batch	批
Binary	二进制
Black box testing	黑盒测试
Bluetooth	蓝牙技术
Boolen algebra	布尔代数
Bottleneck	瓶颈
Breakpoint	断点
Bridge	网桥
Broadband	宽带
Buffer	缓冲区
Bug	缺陷
Bundle	捆绑
Business	业务、商业
Cable	电缆
Cache	高速缓冲存储器
Calculator	计算器
Call back	回调
Catalog	目录
Category	范畴
Certification	认证
Channel	信道
Class diagram	类图
Cleanroom software engineering	净室软件工程
Clipboard	剪贴板

Cohesion	内聚
Collaboration diagram	协作图
Collaboration	协作
Combinatory Mathematics	组合数学
Commerce	商务
Commit	提交
Compact	紧凑的
Compatibility	兼容性
Compile	编译
Compiler	编译器
Component	组件
Composite	复合
Computation	计算
Conceptual design	概念设计
Concurrent	并发的
Confidential	机密的
Configuration	配置
Congestion	拥挤、阻塞
Connection pool	连接池
Connector	连接件
Consistency	一致性
Console	控制台
Constrain	约束
Container	容器
Context	上下文
Coordinate	坐标
Copyright	著作权
Counter	计数器
Coupling	耦合
Cracker	骇客
Critical path	关键路径
Critical section	临界区
Crosscut	横切
Crystal	水晶、水晶方法
Data mining	数据挖掘
Data warehouse	数据仓库
Datagram	数据报
Debug	调试
Decision theory	决策理论
Decision tree	决策树
Decompile	反编译
Decryption	解密
Definition	定义
Delegate	代理、委托
Delegated administration	委托管理
Demo	样本
Demodulation	解调
Dependency	依赖
Deployment	部署
Derive	派生
Descriptor	描述符/描述器
Design by contract	契约式设计
Design pattern	设计模式
Diagnostics	诊断
Digital certificate	数字证书
Digital signature	数字签名
Disassemble	反汇编
Discrete mathematics	离散数学
Divergent	分歧
Dizzy	混乱的
Documentation	文档
Domain model	域模型
Domain-specific	领域相关的
Dot product	点积
Driver	驱动程序
Duplex system	双工系统
Duplex	双工
Durability	持久性
Dynamic	动态
Electronics	电子学
Element	元素
Embedded system	嵌入式系统

Emulation	仿真
Encapsulation	封装
Encryption	加密
Engine	引擎
Entity	实体
Ethernet	以太网
Euclidean space	欧氏空间
Even	偶数、偶校验
Evolutionary	进化的
Exception	异常
Executable	可执行的
Extension	扩展
Extract	提取
Extranet	外联网
Facsimile	传真
Fault tree	错误树
Fault-tolerant	容错
Feasibility	可行性
Feedback	反馈
Field	字段
Filter	过滤
Floppy disk	软盘
Flow chart	流程图
Flow control	流量控制
Foreign key	外键
Format	格式、格式化
Framework	框架
Frame	帧
Frequency	频率
Function overloading	函数重载
Function	功能、函数
Functional testing	功能测试
Fuzzy	模糊的
Game theory	对策论
Gantt chart	甘特图
Gateway	网关
Generative	再生的
Generic programming	泛型编程
Generic	泛型
Geometric	几何的
Global	全局的
Granularity	粒度
Graph theory	图论
Grey box testing	灰盒测试
Grid	网格
Guaranteed delivery	可靠性传输
Hacker	黑客
Handle	句柄
Handwriting recognition	手写识别
Harness	约束
Hashtable	哈希表
Heap	堆
Hierarchical	层次的、体系的
High availability	搞可用性
Hook	钩子
Human factors engineering	人因工程
Hybrid programming	混合编程
Hypermedia	超媒体
Hypertext	超文本
Hypothetical	假定的
Icon	图标
Identifier	标识符
Imagebase	基地址
Increment	增量
Incremental integration testing	组合测试
Infer	推理
Information hiding	信息隐藏
Infrastructure	下部构造、基础的下部组织
Inheritance	继承
Initialize	初始化
Install	安装

Instance	实例
Instruction	指令
Integration testing	集成测试
Integration	集成
Intelligence	智能
Intensity	强调
Interceptor	拦截器
Intermediate	中间的
Internationalization	国际化
Interpret	解释
Intranet	内部网、企业网
Inversion	反转
Invoke	调用
Isolation	孤立性
Isomorphic	同构的
Iterative	迭代
Iterator	迭代器
Join point	连接点
Kernel	内核
Large-scale	大规模的
License	许可证
Life cycle	生命周期
Lifeline	生命线
Linear	线性的
Linearization	线性化
Linguistics	语言学
Liquid-crystal	液晶的
Load testing	负载测试
Load-balanced	负载均衡的
Location	定位
Log	日志
Logics	逻辑学
Macro	宏
Magnetic	磁性的
Maintenance	维护
Managed execution	托管执行
Manual	手册
Mapping	映射规则
Marshalling	编组、封送
Matrix	矩阵
Mechanism	机制
Mentor	导师
Merge	归并
Method	方法
Microwave	微波
Middleware	中间件
Migration	移植
Mirror	镜像、镜子
Modem	调制解调器
Modulation	调制
Module coupling	模块耦合
Module	模块、组件
Monitor	监视器
Motherboard	主板
Multiprogramming	多道程序设计
Multithreading	多线程
Mutation	变异
Namespace	名字空间
Natural language	自然语言
Navigation	定位、航行
Neural network	神经网络
Novice	初学者
Number theory	数论
Numerical computation	数值计算
Open source	开放源代码
Operator	操作符
Optical fiber	光纤
Optical	视力的、光学的
Optimization	优化
Orthogonal	正交
Outsourcing	外包
Over-engineering	过度设计

Overflow	溢出
Overload	重载
Override	覆盖
Package	包
Pair programming	结对编程
Panel	面板
Paradigm	泛例
Parameter	参数
Parity	奇偶
Pattern matching	模式匹配
Peer-to-peer computing	对等计算
Performance testing	性能测试
Performance	性能
Peripheral	外设
Persistence	持久性
Personalize	使个性化
Pipelining	流水线
Pixel	像素
Platform independent	平台无关
Platform invoke	平台调用
Platform	平台
Plug-in	插件
Pointer	指针
Policy	策略
Polymer	聚合体
Polymorphism	多态
Port	端口
Portability	可移植性
Portal	门户
Prepositional logic	命题逻辑
Preprocessor	预处理程
Primary key	主键
Priority	优先权
Probability theory	概率论
Procedure	过程
Process	处理
Processor	处理器
Production	产生式、成果、产品
Profile	框架、轮廓
Projection	投影
Property	属性
Protocol	协议
Prototyping development approach	型化开发方法
Proxy	代理
Pruning node	修剪结点
Pseudocode	伪代码
Quota	定额
Reactor	反应器
Real-time	实时的
Recovery testing	恢复测试
Redundancy	冗余
Refabricate	重构
Reference type	应用类型
Reference	引用
Referential integrity	参照完整性
Reflection	反射
Register	注册
Regular expression	正则表达式
Relational algebra	关系代数
Relational databases model	关系数据库模型
Release	发布
Remote	远程的
Repeater	中继器
Replication	复制
Repository	数据仓库、仓库
Resident	常驻的
Resolution	分辨率、决定的
Responsiveness	响应
Retrieve	检索
Reusability	复用性
Reverse engineering	逆向工程
Robot	机器人

Robust	健壮的
Rollback	回滚
Router	路由器
Sandbox	砂箱
Satellite	人造卫星
Scan	扫描
Scheduling	调度
Schema	模式、结构、方案
Scheme	方案、系统
Script	脚本
Search engine	搜索引擎
Security testing	安全测试
Security	安全性
Segment	段
Semantic	语义的
Semiconductor	半导体
Sensor	传感器
Sequential	顺序的
Serial	串行的
Serialize	串行化
Server cluster	服务器集群
Set theory	集合论
Set-top box	机顶盒
Shading	投影
Shareware	共享软件
Side effect	副作用
Signature	签名
Silicon	硅
Simplex	单工
Simulation	模拟
Simultaneous	同步的
Smart pointer	指针
Sockets layer	套接层
Software reuse	软件复用
Solution	解决方案

Sophisticated	高级的、复杂的
Speech recognition	语音识别
Speech synthesis	语音合成
Spiral model	螺旋模型
Spreadsheet	图表
Spyware	间谍软件
Stack	栈
Standardize	使标准化
Statistical	统计的
Stored procedure	存储过程
Strategy	策略
Stream	流
Stress testing	压力测试
String	串
Stub	存根
Subject	主体
Subnet	子网
Substantial	实质的
Supercomputer	超级计算机
Symbol	符号
Synchronize	使同步
Syntactic	语法的
System analyst	系统分析员
System testing	系统测试
Template	模板
Terminal	终端
Terminology	术语
Tertiary	第三方的
Test case	测试用例
Test Driven development	测试驱动开发
Thread pool	线程池
Thread	线程
Threshold	阈值
Throughput	吞吐量
Time-slicing	时间片
Token	令牌

Top-down programming	自顶向下程序设计
Topology	拓扑（结构）
Tow-way	双向的
Track	追踪
Transaction	事务
Transformation	转换
Transistor	晶体管
Trigger	触发器
Tuple space	元组空间
Unicode	国际双字节编码
Uninstall	卸载
Unit testing	单元测试
Unmarshalling	反编组、拆收
Upward compatible	向上兼容的
Use case	用例
User identity	用户身份认证
Utility	效用、工具
Vacuum tube	真空管
Value chain	价值链
Variable-length array	可变长数组
Variance	变动、协变
Vector	矢量
Velocity	速率
Vibration	震荡
View	视图
Violation	冲突
Virtual memery	虚拟内存
Virtual	虚拟的
Virus	病毒
Visual	可视化的
Wafer	晶片
Waterfall method	瀑布方法
Webservice	Web 服务
White box testing	白盒测试
Workflow	工作流
Workplace	工作区
Workstation	工作站
Worm	蠕虫病毒

12.2.2　常见计算机缩略语

3D	（three dimension）：	三维
ACE	（adaptive communication environment）：	可适配通信软件开发环境
ACM	（association for computing machinery）：	美国计算机学会
ADO	（ActiveX data objects）：	ActiveX 数据对象
ADSL	（asymmetrical digital subscriber line）：	非对称数字用户环路
AI	（artificial intelligence）：	人工智能
AMI	（asynchronous message invocation）：	异步消息
ANSI	（American national standards institute）：	美国国家标准化协会
AOP	（aspect oriented programming）：	面向方面编程
AP	（application plan）：	应用程序规划
API	（application programming interface）：	应用编程接口
ARP	（address resolution protocol）：	地址解析协议
ASCII	（American standard code for Information Interchange）：	美国国家信息交换标准码
ASD	（adaptive software development）：	自适应软件开发
ASP	（active server page）：	动态服务器页技术

ATM	（asynchronous transfer model）：	异步传输模式
B/S	（browser/server）：	浏览器/服务器结构
B2B	（business to business electronic commerce）：	企业对企业的电子商务
B2C	（business to consumer electronic commerce）：	企业对客户的电子商务
C/S	（client/server）：	客户端/服务器结构
CAD	（computer aided design）：	计算机辅助设计
CASE	（computer aided software engineering）：	计算机辅助软件工程
CDMA	（code division multiple access）：	码分多址技术
CGA	（color graphics adapter）：	彩色图形适配器
CIM	（computer –integrated manufacturing）：	计算机集成制造技术
CISC	（complex instruction set computer）：	复杂指令集计算机
CLI	（common intermediate language）：	通用中间语言
CLR	（common language runtime）：	公共语言运行环境
CLS	（common language specification）：	公共语言规范
CMM	（capability maturity model）：	能力成熟度模型
CMMI	（capability maturity model integration）：	能力成熟度模型综合
CMP	（container managed persistence）：	容器管理数据一致性
COM	（component object model）：	组件对象模型
CORBA	（common object request broker architecture）：	公共对象请求代理体系结构
CRC	（cyclic redundancy check）：	循环冗余校验
CRM	（customer relationship management）：	客户关系管理
CSMA/CD	（carrier sense multiple access collision detect）：	载波侦听多路访问/冲突检测
DAO	（data access object）：	数据访问对象
DBA	（database administrator）：	数据库管理员
DBMS	（database management system）：	数据库管理系统
DCE	（distributed computing environment）：	分布式计算机环境
DCOM	（distributed component object model）：	分布式组件对象模型
DFA	（deterministic finite automaton）：	确定有限状态自动机
DFD	（dataflow diagram）：	数据流图
DHTML	（dynamic hypertext markup language）：	动态超文本标记语言
DLL	（dynamic-link library）：	动态链接库
DNS	（domain name System）：	域名系统
DoS	（denial of service）：	拒绝服务攻击
DSDM	（dynamic system development method）：	动态系统开发方法
DSS	（decision support system）：	决策支持系统
ECC	（error correction）：	纠错码

ECO	（enterprise core object）:	企业核心对象
EDI	（electronic data interchange）:	电子数据交换
EJB	（enterprise javabean）:	企业 Javabean
ERD	（entity-relationship diagram）:	实体联系图
ERP	（enterprise resource planning）:	企业资源计划
ES	（expert system）:	专家系统
FAT	（file allocation table）:	文件分配表
FDD	（feature-driven development）:	特征驱动开发
FIFO	（first-in first-out）:	先进先出
FTP	（file transfer protocol）:	文件传输协议
GA	（genetic algorithm）:	遗传算法
GC	（garbage collection）:	内存垃圾收集
GIS	（geographic information system）:	地理信息系统
GPS	（global positioning system）:	全球定位系统
GSM	（global system for mobile communication）:	全球移动通信系统
GUI	（graphics user interface）:	图形用户界面
HTML	（hypertext markup language）:	超文本标记语言标准
HTTP	（hypertext transfer protocol）:	超文本传输协议
IC	（integrated circuit）:	集成电路
ICMP	（Internet control message protocol）:	网际报文控制协议
IDE	（integration development environment）:	集成开发环境
IDS	（intrusion detection system）:	入侵检测系统
IEEE	（institute for electrical and electronic engineers）:	美国电气电子工程师学会
IGMP	（Internet group multicast protocol）:	网际成组多路广播协议
IP	（Internet protocol）:	网际协议
IPC	（interprocess communication）:	进程间通信
IPS	（intrusion prevention system）:	入侵防护系统
ISA	（industry standard organization）:	工业标准化组织
ISDN	（integrated services digital network）:	综合数字业务网
ISO	（international organization for standardization）:	国际标准化组织
ISP	（Internet service provider）:	因特网服务提供商
J2EE	（Java 2 enterprise edition）:	Java 2 企业版
J2ME	（Java 2 micro edition）:	Java 2 袖珍版
J2SE	（Java 2 sdk standard edition）:	Java 2 标准版
JDBC	（Java database connectivity）:	Java 数据库连接
JDK	（Java developer's Kit）:	Java 开发工具包

JDO （Java database object）： Java 数据对象
JPEG （joint photo-graphic experts group）： 联合图像专家组（压缩标准）
JSP （Java server page）： Java 服务器页面技术
JVM （Java virtual machine）： Java 虚拟机
LAN （local-area network）： 局域网
MAC （media access control）： 介质访问控制
MAN （metropolitan-area network）： 城域网
MDA （model driven architecture）： 模型驱动构架
MFC （Microsoft foundation class）： 微软公司 VC++类库名
MIMD （multiple instruction multiple data）： 多指令多数据
MIS （management information system）： 管理信息系统
MOF （managed object format）： 管理的对象格式
MPEG （moving picture experts group）： 运动图像专家组（标准）
MSDN （Microsoft developer network）： 微软开发者网络
MUD （multiple user dimension）： 多人文字角色扮演游戏
MVC （model-view-controller）： 文档-视图-控制模式
NFS （network filing system）： 网络文件系统
OA （office automation）： 办公自动化
OCL （object constraint language）： 对象约束语言
OCR （optical character recognition）： 光学字符识别
ODBC （open database connectivity）： 开放数据库连接
OEM （original equipment manufacture）： 原始设备制造商
OLAP （online analytical processing）： 联机分析处理
OLE （object linking and embedding）： 对象链接和嵌入
OMG （the object management group）： 对象管理组织
OMT （object modeling technique）： 对象建模技术
OO （object oriented）： 面向对象的
OOD （object oriented design）： 面向对象的设计
OOP （object oriented programming）： 面向对象的编程
ORB （object request broker）： 对象请求代理
OSI （open system interconnect reference model）： 开放式系统互联参考模型
OWL （object window library）： 对象窗口库
PCI （peripheral component interconnect）： 外部设备互联
PHP （PHP hypertext preprocessor）： PHP 超文本处理器（语言名，递归定义）
POP3 （post office protocol，Version 3）： 电子邮局协议，版本 3

PSP	（personal software process）：	个体软件过程
QA	（quality assurance）：	质量保证
QoS	（quality of service）：	服务质量
RAD	（rapid application development）：	快速应用程序开发
RAM	（random-access memory）：	随机存储器
RAP	（Internet route access protocol）：	网际路由存取协议
RARP	（reverse address resolution protocol）：	逆向地址解析协议
RDF	（resource description framework）：	资源描述框架
RIP	（routing information protocol）：	路由信息协议
RISC	（reduced instruction set computer）：	精简指令集计算机
RMI	（remote method invocation）：	远程方法调用
ROM	（read-only memory）：	只读存储器
RPC	（remote procedure call protocol）：	远过程调用协议
RPG	（role play games）：	角色扮演游戏
RUP	（Rational unified process）：	瑞理公司软件统一开发过程
SCM	（software configuration management）：	软件配置管理
SDK	（software development Kit）：	软件开发工具包
SMP	（symmetric multi processing）：	对称多处理系统
SMTP	（simple mail transfer protocol）：	简单邮件传输协议
SNMP	（simple network management protocol）：	简单网络管理协议
SOAP	（simple object access protocol）：	简单对象访问协议
SQL	（structured query language）：	结构化查询语言
STL	（standard template library）：	标准模板库
TCP	（transmission control protocol）：	传输控制协议
TSP	（team software process）：	团队软件过程
UDDI	（universal description，discovery and integration）：	统一描述、发现和集成协议
UDP	（user datagram protocol）：	用户数据报协议
UI	（user interface）：	用户界面
UML	（the unified modeling language）：	统一建模语言
UP	（unified process）：	统一软件开发过程
URL	（uniform resource locators）：	通用资源定位符标准
USB	（universal serial bus）：	通用串行总线
VAS	（value-added serve）：	增值服务
VCD	（video compact disc）：	视频光盘
VCL	（visual component library）：	可视化构件库
VGA	（video graphics adapter）：	视频图形适配器

VLAN	（virtual local-area network）：	虚拟局域网
VOD	（video on demand）：	视频点播系统
VPN	（virtual private network）：	虚拟专用网络
VRML	（virtual reality modeling language）：	虚拟现实建模语言
W3C	（world wide web consortium）：	万维网联盟
WAN	（wide-area network）：	广域网
WAP	（wireless application protocol）：	无线应用协议
WCDMA	（wideband code division multiple access）：	多频码分多址技术
WLAN	（wireless local-area network）：	无线局域网
WSDL	（web service description language）：	Web 服务描述语言
WWW	（world wide web）：	万维网
XAML	（extensible application markup language）：	可扩展应用程序标记语言
XML	（extensible markup language）：	可扩展标记语言
XP	（extreme programming）：	极限编程

第 13 章

信息化基础知识

本章主要介绍信息化的基础知识。

13.1 信息与信息化

本节主要介绍信息与信息化的基本概念与特点。

13.1.1 信息的定义及其特性

什么是信息？香农在《通信的数学理论》一文中对“信息”的理解是“不确定性的减少”，由此引申出信息的一个定义：信息是系统有序程度的度量。同年，控制论的创始人维纳在《控制论》一书中指出，“信息就是信息，不是物质也不是能量”。当然，人们还从不同的角度给信息下了定义，据统计，目前信息的定义不下几十种。但是，被人们所普遍接受的大概还是香农的定义，因为香农不但给出了信息的定义，而且还给出了信息的定量描述，并确定信息量的单位为比特（bit）。1 比特的信息量，在变异度为 2 的最简单情况下，就是能消除非此即彼的不确定性所需要的信息量。香农把热力学中的熵引入信息论。在热力学中，熵是系统无序程度的度量，而信息与熵正好相反，信息是系统有序程度的度量，因而，表现为负熵。

13.1.2 信息化

1．信息化的定义

信息化（Informationalization）一词是由日本学者在 20 世纪 70 年代提出的，迄今为止，还没有一个广为接受和认可的权威定义。所谓信息化，可以认为是现代信息技术与社会各个领域及其各个层面相互作用的动态过程及结果。在这一相互作用过程中，信息技术自身和整个社会都发生着质的变化。其中，社会的质的变化主要表现为信息资源开发和应用及知识生产力迅速提高的结果。信息化是与当代信息革命、信息社会相关联的，信息化不同于工业化，工业化是信息化的基础，信息化可以促进工业化的进程；信息化不等同于现代化，在现代的时代背景下，信息化是现代化的目标之一；信息化不等于自动化，传统的自动化设备是以物质能源来驱动的，而对于信息化设备而言，信息不仅是处理对象，而且是信息系统的资源。

从本质上看，信息化应该是以信息资源开发利用为核心，以网络技术、通信技术等高科技为依托的一种新技术扩散的过程。作为这一过程的结果，它最终将会引起整个产业结构的变化。

2．信息化的内容

信息化是一个非常宽泛的和宏观的概念，而当人们谈到信息化时总是具体的和有针对性的。关于信息化的内容，一般来说，其针对性非常强。几年前，我国国家信息化管理部门列出了国家信息化体系的六个要素，可以作为区域信息化、行业信息化、企业信息化等的参考。

一是信息资源。信息和材料、能源共同构成经济和社会发展的三大战略资源。我国信息资源极其丰富，但开发利用的程度较低，远远落后于需要。因此，开发和利用信息资源是我国信息化的关键一环和决定性的一环。

二是信息网络。信息网络是信息资源开发、利用的基础设施，信息网络包括计算机网络、电信网、电视网等。信息网络在国家信息化的过程中将逐步实现三网融合，并最终做到三网合一。

三是信息技术应用。信息技术应用是国家信息化中十分重要的要素，它直接反映了效率、效果和效益。

四是信息产业。信息产业是信息化的物质基础。信息产业包括微电子、计算机、电信等产品和技术的开发、生产、销售，以及软件、信息系统开发和电子商务等。从根本上来说，国家信息化只有在产品和技术方面拥有雄厚的自主知识产权，才能提高综合国力。

五是信息化人才。人才是信息化的成功之本，而合理的人才结构更是信息化人才的核心和关键。合理的信息化人才结构要求不仅要有各个层次的信息化技术人才，还要有精干的信息化管理人才、营销人才，法律、法规和情报人才。在信息化人才中有一种人才最为重要，那就是系统分析师。系统分析师既是信息化的技术人才，同时又是经营管理人才，是一种复合型人才。而 CIO（首席信息官）又是系统分析师队伍的领军人物，是企业最高管理层的重要成员。

六是信息化政策、法规、标准和规范。信息化政策和法规、标准、规范是国家信息化快速、有序、健康和持续发展的保障。

13.1.3　组织对信息化的需求

组织对信息化的需求是组织信息化的原动力，它决定了组织信息化的价值取向和成果效益水平。而需求本身又是极为复杂的，它不是组织中各个部门对计算机技术和功能需求的简单罗列，也不是对信息系统和信息项目需求的简单叠加，而是一个系统的、多层次的目标体系。

一般说来，信息化需求包含三个层次，即战略需求、运作需求和技术需求。

1．战略需求

组织信息化的目标是提升组织的竞争能力，为组织的可持续发展提供一个支持环境。从某种意义上来说，信息化对组织不仅仅是服务的手段，也不仅仅是实现现有战略的辅助工具；信息化可以把组织战略提升到一个新的水平，可以为组织带来新的发展契机。特别是对于企业，信息化战略是企业竞争的基础。

例如，沃尔玛从一个小杂货店到现今稳坐世界 500 强的第一把交椅，一个重要原因就是借力于信息化。1969 年，世界才开始进入计算机时代，沃尔玛就租用了一台 IBM360 计算机用于配送中心的存货控制。到了 20 世纪 80 年代初，沃尔玛在世界上第一个发射了企

业自己的人造卫星，用于企业内部通信。据说，沃尔玛的电子通信系统是全美最大的民用系统，甚至超过了电信业巨头美国电报电话公司。沃尔玛应用强大的通信和计算机处理能力，大大降低了其营业成本，保证了企业以最低的价格、最全的品种为顾客服务。

2．运作需求

世界著名管理咨询公司 —— 麦肯锡公司在为各种组织（政府机构、企业等）做咨询时，无论面对多么复杂的咨询项目都能化繁为简，从而取得成功。面对复杂的问题，麦肯锡总会归纳成三个大问题：在哪里？去哪里？怎么去？然后，再对三个大问题进行细化。在哪里，就是要搞清现状，包括组织的发展历史和内外部环境，以及组织的绩效、实力等；去哪里，就是要回答组织的战略目标是什么；怎么去，就是要回答如何运作才能实现组织的战略目标。

由此看来，组织信息化的运作需求是组织信息化需求非常重要且关键的一环，它包含三方面的内容：一是实现信息化战略目标的需要，因为一个战略目标制定以后，必须通过实际运作来实现，而实际运作的过程中，会不断提出新的信息化需求；二是运作策略的需要，组织信息化策略选择是实现信息化战略的支持力量，比如，是自主开发还是委托开发，在什么时机进行开发，如何培训，对相关产品如何选型等；三是人才培养的需要，组织信息化的一个关键功能是就是人才的培养。

3．技术需求

信息化技术是组织发展的一个支撑环境，由于一些组织的信息化建设进行了相当长的时间，一些系统已不满足于目前的需求，有的甚至形成了许多信息孤岛等，这些问题在信息技术层面上对系统的完善、升级、集成和整合提出了需求。也有的组织原来基本上没有上大的信息系统项目，有的也只是一些单机应用，做一些文字处理工作，或是上网浏览一下有用信息。这样的组织的信息化需求，一般要从头开发新的系统。

组织的三个层次的需求并不是相互孤立的，而是有着内在的联系。信息化需求的获取是一个自上而下的过程，需要对这些需求进行综合分析，才能把握组织对信息化建设的方向。

一个组织就是一个系统，并且是一个复杂的系统。组织的各层次的信息化需求之间并不是互相独立、互不相关的，而是存在着有机的内在联系。搞清不同层次需求之间的关系对于组织信息化的实施非常重要，其实，它就是信息化所要解决的问题。

各层次信息化需求之间的逻辑关系包括的因果关系、依赖关系、主辅关系、协同关系等。

实现组织信息化是需要资源的，包括人力、物力和财力，以及时间和精力等资源，而任何一个组织所拥有的资源总是有限的，不可能满足所有的需求。在这种情况下，一个组织的信息化应该遵循“总体规划，分步实施”的原则，在多方面、多层次的需求中，首先考虑那些关键的、主要的，并且是资源条件允许的需求。另一方面，在组织信息化基础比较薄弱，员工对信息化的认识和技术水平较低的情况下，如果能从相对比较容易实施和产生效果的环节切入，使组织能在短时间内实实在在地体会到信息化所带来的效果，这对组织信息化的推进非常有好处。

13.2 政府信息化与电子政务

本节主要介绍政府信息化与电子政务。

13.2.1　政府信息化的概念、作用及意义

1．政府信息化的含义

政府信息化，就是传统政府向信息化政府演变的过程。具体来说，政府信息化就是应用现代信息技术、网络技术和通信技术，通过信息资源的开发和利用来集成管理和服务，从而提高政府的工作效率、决策质量、调控能力，并节约开支，改进政府的组织结构、业务流程和工作方式，全方位地向社会提供优质、规范、透明的管理和服务。

这个定义包含三个方面的内容：第一，政府信息化必须借助于信息技术和网络技术，离不开信息基础设施和软件产品；第二，政府信息化是一个系统工程，它不仅是与行政有关部门的信息化，还包括立法、司法部门及其他一些公共组织的信息化；第三，政府信息化并不是简单地将传统的政府管理事务原封不动地搬到互联网上，而是要对已有的组织结构和业务流程进行重组或再造。

这里需要说明的是，政府信息化的主要内容是电子政务。因此，在大多数情况下，电子政务可以作为政府信息化的同义语来使用。

2．政府信息化的作用和意义

政府信息化的作用和意义如下。

一是能够提高政府的行政效率和效用。政府机构是否能正确地履行职能依赖于充分和准确的信息；政府是社会发展的导航者，需要准确把握社会对政府的需求及其变化，从而才能准确地做出战略和策略选择；政府是社会问题的治理者，这要依赖于对各方面信息的深入分析、正确判断和准确把握；政府是市场失灵的矫正者，矫正市场失灵的前提是最大程度地掌握信息。因此，充分的信息和信息网络是现代政府的神经系统。而政府信息化无疑对政府搜集处理信息提供了极大的便利。具体说来，政府信息化对政府管理乃至社会信息化具有非常积极的作用和重要意义。

二是对于社会信息化起到示范作用。政府信息化是社会信息化的重要组成部分，同时，由于政府是公共品的提供者，因而政府信息化对社会信息化具有很好的示范作用。

三是能够提高政府的服务质量。政府无疑是社会上最大的信息资源拥有者和使用者之一，政府管理和行政的过程就是信息收集、处理和存储的过程。政府信息化可使政府运用所获取和掌握的信息，对社会问题和群众的需求做出准确的分析和判断，从而提高公共品的质量。同时，促进信息的流通和共享。

四是可以使人力资源得到优化配置。人才不仅是企业的重要资源，更是政府的主要资源。政府的行政和管理主要靠人才，靠人才的知识和智力。政府信息化使得政府工作人员面临着更多的挑战和压力，从而促使他们不断地去学习，不断地更新自己的知识和技能。同时，网络的发展也为政府工作人员提供了学习提高的极好的环境和平台，为他们的素质提高创造了良好的条件。

五是有利于反腐倡廉。网络使得信息的传递做到快速和及时，使信息的发布和反馈能够及时或实时，为政府的动态管理提供了可能；政府上网后，政府通过网络宣传各种政策，扩大了服务职能，提高了办事效率，增加了政府工作的透明度，这有利于遏制进而消除官僚主义；政府信息化的发展使政府的管理呈现了动态性和透明性，有利于有关部门和人民群众的监督，有利于消除官僚主义、文牍主义，有利于反腐倡廉。

六是做到信息共享。各级政府掌握着大部分的社会、经济文化信息及全部的政策和法律信息。政府信息化使得信息资源不再只是储存于仓库的档案或柜中的资料，而是成为创造价值的富有生命力的社会资源。也只有在政府信息化的前提下，信息共享才不再是一句空话。

七是有利于节约开支。政府信息化必将对政府组织结构和运作方式产生冲击。政府信息化使传统的部门组织朝着网络组织的方向发展，打破了职位、层级、部门的限制，促进政府组织和职能的整合，使政府的业务流程更加简明、畅通，从而可以使政府节约人力、物力和财力资源，减少成本开支，提高办事效率。

13.2.2　我国政府信息化的历程和策略

20 世纪 90 年代以来，伴随着信息技术、特别是网络技术的飞速发展，信息化成为各国普遍关注的一个焦点。在国家信息化体系建设中，政府信息化又成为整个信息化中的关键。

1．我国政府信息化的发展历程

我国政府信息化最早起始于 20 世纪 80 年代末期“中国国家经济信息系统”的建设和运行。

当时，我国计划经济体制正在开始向市场经济体制转轨，社会发展对于经济信息的需求非常强烈。在这样的情况下，建设国家经济信息系统正是适合了国家和社会的多种需求。国家经济信息系统包括着重为国家宏观经济服务的主系统，以及部门各个行业的专业经济信息系统在内的全国系统。同时，组建了国家经济信息中心作为国家经济信息系统的重要组成部分。国家经济信息中心是整个国家经济信息系统设计、规划、实施和技术协调的承担单位，是政府对全国经济信息事业的归口管理单位，它还负责经济信息政策的研究和经济信息系统技术规范和标准的制定。

国家经济信息系统不但为现今的政府信息化和电子政务提供了丰富的经验积累，也为企业信息系统的建设和运行起到了很好的示范作用。

20 世纪 90 年代，随着信息技术的飞速发展和广泛的应用，我国政府信息化也得到了长足的发展，其中最主要的成果如下。

一是以“金”字头为代表的多项信息工程项目取得了突破性进展。从 1993 年起，我国开始实施金桥、金关、金卡和金税等重大信息化工程。金桥工程是直接为国家宏观经济调控和决策服务的，通过建设政府的专用基础通信网，实现政府之间的相互联接，形成一个连接全国各省市区、400 多个城市，与几十个部委互联的专用网。金关工程主要是为提高外贸及相关领域的现代化管理和服务水平而建立的信息网络系统。到 1999 年，已实现了银行、外汇管理机构及海关的计算机联网，在关税管理中发挥了重要作用。金卡工程是推动银行卡跨行业务的联营工作，现已取得了重要进展。金税工程的首期工程已经完成，主要是建立税务系统的增值税专用发票计算机稽核系统。

二是政府上网工程初具规模。在“金”字系统工程取得重大进展的同时，从 1999 年起，在全国普遍实行了政府上网工程。到目前为止，全国绝大多数县级以上政府都实现了电子政务。

三是一些地区、部门在政府信息化方面已取得了显著成效。在中央的大力倡导下，各地在推动政府信息化方面正在全面发展。

2．“十二五规划”信息化战略

1）“十二五”信息化规划目标

“十二五”信息化规划要通过“两化融合”提升传统产业竞争力，大力发展信息产业等新兴产业，提高政府和社会大众的信息化应用水平，全面支撑工业强国和信息社会的发展目标。

《国家信息化的战略目标（2006-2020）》明确制定了我国到 2020 年应该达到的一系列重要目标为：

- 综合信息基础设施基本普及。
- 信息技术自主创新能力显著增强。
- 信息产业结构全面优化。
- 国家信息安全保障水平大幅提高。
- 国民经济和社会信息化取得明显成效。
- 新型工业化发展模式初步确立。
- 国家信息化发展的制度环境和政策体系基本完善。
- 国民信息技术应用能力显著提高。
- 为迈向信息社会奠定坚实基础。

2）“十二五”信息化规划主要任务

信息化是充分利用信息技术，开发利用信息资源，促进信息交流和知识共享，提高经济增长质量，推动经济社会发展转型的历史进程。信息化发展对中国经济、社会具有十分重大的影响。我国信息化的五大应用领域如下：经济领域的信息化，包括农业信息化、服务业信息化、两化融合、信息产业等；社会领域的信息化，包括民生、公共卫生、劳动保障等；政务领域的信息化，包括政府办公、对外服务等；文化领域的信息化，包括图书、档案、文博、广电、网络治理等；军事领域的信息化，包括装备、情报、指挥、后勤等。国民经济和社会发展信息化“十二五”规划主要关注经济、社会和政务领域的信息化，部分地区包含文化领域信息化。

13.2.3　电子政务的概念、内容和技术形式

1．电子政务的概念

20 世纪 90 年代，信息技术迅猛发展，特别是伴随着互联网技术的普及应用，电子政务的概念便应运而生了。电子政务一出现，就成为信息化的最重要的领域之一。根据联合国教科文组织在 2000 年对 62 个国家（39 个发展中国家，23 个发达国家）所进行的调查，89%的国家都在不同程度上着手推动电子政务的发展，并将其列为国家级的重要事项。事实上，电子政务已经迅速地列入了所有工业化国家的政治日程。

电子政务实质上是对现有的、工业时代形成的政府形态的一种改造，即利用信息技术和其他相关技术，来构造更适合信息时代政府的组织结构和运行方式。现有的政府组织形态是工业革命的产物，与工业化的行政管理的需求和技术经济环境相适应，已经存在了 200 年以上。随着网络时代和网络经济的来临，管理正由传统的金字塔模式走向网络模式。政府的组织形态也必然由金字塔式的垂直结构向网状结构转变，从而减少管理的层次，以各种形式通过网络与企业和居民建立直接的联系。因此，电子政务的发展过程实质上是对原

有的政府形态进行信息化改造的过程，通过不断地摸索和实践，最终构造出一个与信息时代相适应的政府形态。

在信息时代，就像管理信息系统是管理企业必备的手段一样，电子政务已经成为国民经济信息化不可或缺的一环。信息化使许多政府原来不可能做到的事情不仅可以做到，而且可以做得更快、更好，帮助政府实现对国家的有效管理。今天，无论经济与社会的发展或者国家安全的保障，都不能没有电子政务的支持。

电子政务的发展对我国的经济和社会发展，特别是信息产业的发展将有着十分重要的影响。电子政务的发展还将对我国各行各业信息化的发展，包括电子商务和电子社区起着示范作用。

2．电子政务的内容

在社会中，与电子政务相关的行为主体主要有三个，即政府、企（事）业单位及居民。因此，政府的业务活动也主要围绕着这三个行为主体展开，即包括政府与政府之间的互动；政府与企、事业单位，尤其是与企业的互动；政府与居民的互动。在信息化的社会中，这三个行为主体在数字世界的映射，构成了电子政务、电子商务和电子社区三个信息化的主要领域。电子商务在经历了一个发展热潮之后，目前正在向一个新的、更扎实的阶段发展；电子政务则是当前全球关注的热点，正在形成一个发展的热潮。

政府与政府，政府与企（事）业，以及政府与居民之间的互动构成了下面五个不同的却又相互关联的领域。

1）*政府与政府*（G2G）

政府与政府之间的互动包括首脑机关与中央和地方政府组成部门之间的互动；中央政府与各级地方政府之间的互动；政府的各个部门之间的互动；政府与公务员和其他政府工作人员之间的互动。这个领域涉及的主要是政府内部的政务活动，包括国家和地方基础信息的采集、处理和利用，如人口信息、地理信息、资源信息等；政府之间各种业务流所需要采集和处理的信息，如计划管理、经济管理、社会经济统计、公安、国防、国家安全等；政府之间的通信系统，包括各种紧急情况的通报、处理和通信系统；政府内部的各种管理信息系统，如财务管理、人事管理、公文管理、资产管理、档案管理等；各级政府的决策支持系统和执行信息系统等。

2）*政府对企业*（G2B）

政府面向企业的活动主要包括政府向企（事）业单位发布的各种方针、政策、法规、行政规定，即企（事）业单位从事合法业务活动的环境，包括产业政策、进出口、注册、纳税、工资、劳保、社保等各种规定；政府向企（事）业单位颁发的各种营业执照、许可证、合格证、质量认证等。“政府对企业”的活动实质上是政府向企业提供的各种公共服务，如构造一个良好的投资和市场环境，维护公平的市场竞争秩序，协助企业，特别是中小企业的发展，帮助企业进入国际市场和加入国际竞争，以及提供各种各样政府信息的服务等。

3）*政府对居民*（G2C）

政府对居民的活动实际上是政府面向居民所提供的服务。政府对居民的服务首先是信息服务，让居民知道政府的规定是什么，办事程序是什么，主管部门在哪里，以及各种关于社区保安和水灾、火灾、天灾等与公共安全有关的信息，户口、各种证件和牌照的管理等政府面向居民提供的各种服务。政府对居民的服务还包括各公共部门如学校、医院、图

书馆、公园等面向居民的服务。

4）企业对政府（B2G）

企业面向政府的活动包括企业应向政府缴纳的各种税款，按政府要求应该填报的各种统计信息和报表，参加政府各项工程的竞、投标，向政府供应各种商品和服务，以及就政府如何创造良好的投资和经营环境，如何帮助企业发展等提出企业的意见和希望，反映企业在经营活动中遇到的困难，提出可供政府采纳的建议，向政府申请可能提供的援助等。

5）居民对政府（C2G）

居民对政府的活动除包括个人应向政府缴纳的各种税款和费用，按政府要求应该填报的各种信息和表格，以及缴纳各种罚款外，更重要的是开辟居民参政、议政的渠道，使政府的各项工作不断得以改进和完善。政府需要利用这个渠道来了解民意，征求群众意见，以便更好地为人民服务。此外，报警服务（盗贼、医疗、急救、火警等）即在紧急情况下居民需要向政府报告并要求政府提供的服务，也属于这个范围。

当前，世界各国电子政务的发展就是围绕着上述五个方面展开的，其目标除不断地改善政府、企业与居民三个行为主体之间的互动，使其更有效、更友好、更精简、更透明和更有效率之外，更强调在电子政务的发展过程中对原有的政府结构及政府业务活动组织的方式和方法等进行重要的、根本的改造，从而最终构造出一个信息时代的政府形态。

13.2.4 电子政务的应用领域

在推动电子政务的过程中，应用领域的确定和选择是一个十分关键的问题。按照我国的国情，在确定电子政务的应用中，既要考虑未来电子政务的发展，也要从实际出发，选择好重点和突破点。我国电子政务的应用领域可以集中在以下六个方面，在具体实施中，则要根据管理的实际，有所选择，确定突破的重点，以滚动式的方式推进电子政务的发展。

一是面向社会的应用。主要包括：政府通过自己的网站向社会发布信息，为社会公众提供查询服务；对于社会向政府传递的各类信息，如信访、建议、反馈等的处理服务；各类公共服务性业务的信息发布和实施，如工商管理、税务管理、保险管理、城建管理等。

二是政府部门之间的应用。主要包括：各级政府间的公文信息审核、传递系统；各级政府间的多媒体信息应用平台，如视频会议、多媒体数据交换等；同级政府间的公文传递、信息交换。

三是政府部门内部的各类应用系统。主要包括：政府内部的公文流转、审核、处理系统；政府内部的各类专项业务管理系统，如日程安排、会议管理、机关事务管理等；政府内部面向不同管理层的统计、分析系统。

四是涉及政府部门内部的各类核心数据的应用系统。主要包括：机要、秘密文件及相关管理系统；领导事务管理系统，如日程安排等；涉及重大事件的决策分析、决策处理系统；涉及国家重大事务的数据分析、处理系统。

五是政府电子化采购，也就是政府的电子商务。

六是电子社区，通过信息化手段为基层群众提供各种便民服务。

13.3 企业信息化与电子商务

本节主要介绍企业信息化与电子商务。

13.3.1 企业信息化的概念、目的、规划、方法

1．企业信息化的概念

企业信息化是指企业以业务流程的优化和重构为基础，在一定的深度和广度上利用计算机技术、网络技术和数据库技术，控制和集成化管理企业生产经营活动中的各种信息，实现企业内外部信息的共享和有效利用，以提高企业的经济效益和市场竞争力。

如果从动态的角度来看，企业信息化就是企业应用信息技术及产品的过程，或者更确切地说，企业信息化是信息技术由局部到全局，由战术层次到战略层次向企业全面渗透，运用于流程管理、支持企业经营管理的过程。这个过程表明，信息技术在企业的应用，在空间上是一个由无到有、由点到面的过程；在时间上具有阶段性和渐进性，起初是战术阶段，经过逐步深化，发展到战略阶段；信息化的核心和本质是企业运用信息技术，进行隐含知识的挖掘和编码化，进行业务流程的管理。企业信息化的实施，一般来说，可以沿两个方向进行，一是自上而下，必须与企业的制度创新、组织创新和管理创新结合；二是自下而上，必须以作为企业主体的业务人员的直接受益和使用水平逐步提高为基础。

2．企业信息化的目的

就一般意义而言，企业信息化的目的就是要建立一个整体上相当于人的神经系统的数字神经系统。这种数字神经系统，使得企业具有平稳和有效的运作的能力，对紧急情况和机会做出快速反应，为企业内外部用户提供有价值的信息，以提高企业的核心竞争力。

企业要应对全球化市场竞争的挑战，特别是大型企业要实现跨地区、跨行业、跨所有制、跨国经营的战略目标，要实施技术创新战略、管理创新战略和市场开拓战略，要将企业工作重点转向技术创新、管理创新和制度创新的方向上来，信息化是必然选择和必要手段。企业信息化涉及对企业管理理念的创新，管理流程的优化，管理团队的重组和管理手段的革新。

首先，技术创新。现实的情况是：一方面，我国企业能够拥有并掌握的技术创新成果甚少，相关信息闭塞；另一方面，又有大量的技术开发成果被沉淀和搁置，造成惊人的浪费。对此，必须运用信息技术，通过在生产工艺设计、产品设计中计算机辅助设计系统的应用，通过互联网及时了解和掌握创新的技术信息，才能加快技术向生产的转化。另外，生产技术与信息技术相结合，能够大幅度地提高技术水平和产品的竞争力，比如，信息技术与洗衣机生产相结合，就生产出了自动洗衣机，增加了附加价值。

其次，管理创新。管理是一门科学，实施管理必须学习和掌握科学的方法。按照市场发展的要求，要对企业现有的管理流程重新整合，从作为管理核心的财务、资金管理，转向技术、物资、人力资源的管理，并延伸到企业技术创新、工艺设计、产品设计、生产制造过程的管理，进而还要扩展到客户关系管理、供应链的管理乃至发展为电子商务。实现这样的管理目标，就必须借助信息技术，发挥计算机的信息采集、存储功能和网络的传递与共享功能。

再次，制度创新。在建立现代企业制度的过程中，信息化起着重要作用。特别是在由计划经济体制向市场经济体制转轨的过程中，赋予企业信息化一系列特殊的使命，那些不

适应企业信息化的管理体制、管理机制和管理制度必须得到创新。同时，通过计算机网络系统管理，建立起明确的岗位责任和精准的监管体系；借助互联网获取更全面、更系统、更及时的信息，彻底改变企业一直沿用的计划经济的资源分配方式和管理方式，注重市场信息的分析和研究，提供准确及时的决策信息；应用科学的方法实施管理。因此，建立在计算机网络技术基础上的管理才更科学、更有效。我们在倡导企业技术改造、技术创新的同时，还应当倡导企业加快管理改造和管理创新。

3．企业信息化的规划

企业信息化一定要建立在企业战略规划基础之上，以企业战略规划为基础建立的企业管理模式是建立企业战略数据模型的依据。

企业信息化就是技术和业务的融合。这个“融合”并不是简单地利用信息系统对手工的作业流程进行自动化，而是需要从三个层面来实现。

首先，企业战略的层面。在规划中必须对企业目前的业务策略和未来的发展方向进行深入分析。通过分析，确定企业的战略对企业内外部供应链和相应管理模式，从中找出实现战略目标的关键要素，分析这些要素与信息技术之间的潜在关系，从而确定信息技术应用的驱动因素，达到战略上的融合。

其次，业务运作层面。针对企业所确定的业务战略，通过分析获得实现这些目标的关键业务驱动力和实现这些目标的关键流程。这些关键流程的分析和确定要根据他们对企业价值产生过程中的贡献程度来确定。关键的业务需求是从那些关键的业务流程的分析中获得的，它们将决定未来系统的主要功能。这一环节非常重要，因为信息系统如果能够与这些直接创造价值的关键业务流程相融合，这对信息化投资回报的贡献是非常巨大的，也是信息化建设成败的一个衡量指标。

再次，管理运作层面。虽然这一层面从价值链的角度上来说，属于辅助流程，但它对企业的日常管理的科学性、高效性非常重要。另外，在企业战略层面的分析中，可以获得适应企业未来业务发展的管理模式，这个模式的实现离不开信息技术的支撑。所以，在管理运作层面的规划上，除提出应用功能的需求外，还必须给出相应的信息技术体系，这些将确保管理模式和组织架构适应信息化的需要。

企业信息化规划的重要性不言而喻，但要防止一种倾向，就是把信息化规划片面地理解为信息技术规划，这样的观念是有害的。

企业战略数据模型分为数据库模型和数据仓库模型。数据库模型用来描述日常事务处理中的数据及其关系；数据仓库模型则描述企业高层管理决策者所需的信息及其关系。在企业信息化过程中，数据库模型是基础，一个好的数据库模型应该客观地反映企业生产经营的内在联系。数据库是办公自动化、计算机辅助管理系统、开发与设计自动化、生产过程自动化、Intranet 的基础和环境。

信息技术和网络技术都在飞速发展，企业信息化是多种类、多层次信息系统建设、集成和应用的过程，因而不是一蹴而就的事情，需要结合企业的实际，全面规划，分步实施。

4．企业信息化方法

企业信息化建设是一项系统工程，不是单元技术的改造，它要涉及企业的方方面面，也就是会涉及企业所处的“生态系统”，个别单位或部分业务的信息化并不能代表整个企业的信息化。企业信息化建设与其说是一场技术变革，还不如说是对企业的经营管理和业务

流程的一次革命，它借助于先进的信息技术和网络技术的价值链进行重构。同时，企业信息化是一个不断发展、变化的过程，它没有终点，至少目前还看不到终点。企业信息化随着管理理念、信息技术和网络技术的发展而发展，是一个螺旋式上升的过程。而在这个过程中，企业使用什么方法来实现信息化，就成为一个事关成败的大问题。

这里需要指出的是，企业信息化方法并不同于信息系统建设方法，这是因为信息系统建设方法是一个具体的信息项目建设的方法，而企业信息化方法是整个企业实现信息化的方法，因此，企业信息化方法要比信息系统建设方法层次更高、涉及面更广。

经过二三十年的发展，人们已经总结出了许多非常实用的企业信息化方法，并且还在探索新的方法。这里只简单介绍几种常用的企业信息化方法。

1）业务流程重构方法

在 20 世纪 90 年代初，美国学者哈默和钱佩在其著作《企业重构》中系统地提出了企业业务流程重构的思想，对美国以至于世界范围内的企业界产生了很大的影响，一时间企业业务流程重构形成了浪潮。

企业业务流程重构的中心思想是，在信息技术和网络技术迅猛发展的时代，企业必须重新审视企业的生产经营过程，利用信息技术和网络技术，对企业的组织结构和工作方法进行“彻底的、根本性的”重新设计，以适应当今市场发展和信息社会的需求。

现在，业务流程重构已经成为企业信息化的重要方法。特别是长期受计划经济体制影响的企业，采用业务流程重构方法来实现企业信息化更有现实意义。

2）核心业务应用方法

任何一个企业，要想在市场竞争的环境中生存发展，都必须有自己的核心业务，否则必然会被市场所淘汰。当然，不同的企业其核心业务是不同的。比如，一个石油生产企业，原油的勘探开发生产就是它的核心业务。围绕核心业务应用计算机技术和网络技术是很多企业信息化成功的秘诀。比如，联想集团用了 10 年的时间，在核心业务流程上应用计算机技术和网络技术，真正实现了业务集成和信息共享，取得了举世瞩目的业绩。

3）信息系统建设方法

对于大多数企业来说，建设信息系统是企业信息化的重点和关键。因此，信息系统建设成了最具普遍意义的企业信息化方法。

4）主题数据库方法

主题数据库就是面向企业业务主题的数据库，也就是面向企业核心业务的数据库。有些企业，特别是大型企业，其业务数量浩大，流程错综复杂。在这样的企业里，建设覆盖整个企业的信息系统往往很难成功，但是，各个部门的局部开发和应用又有很大弊端，会造成系统分割严重，形成许多“信息孤岛”，造成大量的无效或低效投资。

在这样的企业里，应用主题数据库方法推进企业信息化无疑是一个投入少、效益好的方法。例如，对于一个油田企业来说，勘探开发无疑是它的核心业务，有一个大型油田企业，在十几年前，就投入巨大的人力、物力和财力开发“勘探开发数据库”。经过十几年的努力，目前，该数据库已经积累了 GB 级的数据，对企业生产经营发挥了巨大的作用，取得了巨大的经济效益。

5）资源管理方法

资源是企业生存发展的根本保证，一个企业如果离开了资源，那它将一天也活不下去。而资源又包括很多类型，如人力资源、物力资源等；同时，资源又可分为内部资源和外部资源。管理好企业的资源大概是企业管理的永恒主题。

计算机技术和网络技术的应用为企业资源管理提供了强大的能力。因此，资源管理方法也就成了企业信息化的重要方法。

目前，流行的企业信息化的资源管理方法有很多，最常见的有 ERP（企业资源计划）、SCM（供应链管理）等。

6）人力资本投资方法

人力资本的概念是经济学理论发展的产物。人力资本与人力资源的主要区别是人力资本理论把一部分企业的优秀员工看作一种资本，能够取得投资收益。

人力资本投资方法特别适用于那些依靠智力和知识而生存的企业，例如，各种咨询服务、软件开发等企业。

13.3.2　企业资源规划（ERP）的结构和功能

1．ERP 的概念

企业资源计划（Enterprise Resources Planning，ERP）是一种融合了企业最佳实践和先进信息技术的新型管理工具。它扩充了 MIS、MRPⅡ的管理范围，将供应商和企业内部的采购、生产、销售及客户紧密联系起来，可对供应链上的所有环节进行有效管理，实现对企业的动态控制和各种资源的集成和优化，提升基础管理水平，追求企业资源的合理高效利用。ERP 是由美国 Gartner Group 于 20 世纪 90 年代初首先提出的。ERP 实质上仍然以制造资源计划（Manufacturing Resources Planning，MRPⅡ）为核心，但 ERP 至少在两方面实现了拓展，一是将资源的概念扩大，不再局限于企业内部的资源，而是扩大到整个供应链条上的资源，将供应链内的供应商等外部资源也作为可控对象集成进来；二是把时间也作为资源计划的最关键的一部分纳入控制范畴，这使得决策支持系统（DSS）被看作 ERP 不可缺少的一部分，将 ERP 的功能扩展到企业经营管理中的半结构化和非结构化决策问题。因此，ERP 被认为是顾客驱动的、基于时间的、面向整个供应链管理的制造资源计划。

ERP 的概念对应于管理界、信息界、企业界不同的表述要求，“ERP”分别有着它特定的内涵和外延。对于企业来说，要理解 ERP，首先要明确什么是“企业资源”。简单地说，“企业资源”是指支持企业业务运作和战略运作的事物，既包括人们常说的人、财、物，也包括人们没有特别关注的信息资源；同时，不仅包括企业的内部资源，还包括企业的各种外部资源。因此，ERP 就是一个有效地组织、计划和实施企业的内外部资源的管理系统，它依靠 IT 的手段以保证其信息的集成性、实时性和统一性。

2．ERP 的结构

ERP 是一个层次结构，可分为三个层次，即管理思想、软件产品和管理系统。

1）ERP 的管理思想

ERP 最初是一种基于企业内部“供应链”的管理思想，在 MRPⅡ的基础上扩展了管理范围，给出了新的结构。它的基本思想是将企业的业务流程看作一个紧密连接的供应链，将企业内部划分成几个相互协同作业的支持子系统，如财务、市场营销、生产制造、质量

控制、服务维护、工程技术等。最早采用这种管理方式的是制造业，当时主要考虑的是企业的库存物料管理，于是产生了MRP（物料需求计划）系统，同时企业的其他业务部门也都各自建立了信息管理系统，诸如会计部门的计算机账务处理系统、人事部门的人事档案管理系统等，而这些系统早期都是相互独立，彼此之间缺少关联，形成信息孤岛，不但没有发挥IT功能和作用，反而造成了企业管理的管理环节和管理部门的重复和不协调。

在这种情况之下，MRPⅡ应运而生。它围绕着“在正确的时间制造和销售正确的产品”这样一个中心目标，将企业的内外部资源进行集中管理。在一定意义上，ERP可以说是MRPⅡ的一个扩展。第一，它将系统的管理核心从“在正确的时间制造和销售正确的产品”转移到了“在最佳的时间和地点，获得企业的最大增值”；第二，基于管理核心的转移，其管理范围和领域也从制造业扩展到了其他行业和企业；第三，在功能和业务集成性方面，它都有了很大加强，特别是商业智能的引入使得以往简单的事物处理系统变成了真正智能化的管理控制系统。

2）软件产品

随着应用的深入，作为ERP的载体——软件产品，也在向更高的层次发展，已经经历了三个阶段，最初，ERP就是一个软件开发项目。这时的软件产品一般来说，费用高，耗时长，而且项目可控性很差，出现了所谓ERP成功率低的结果。后来，ERP产品发展成模块化，这时，大大地提高了软件开发效率，但是，由于是产品导向，出现了削足适履的现象，因而这时ERP的成功率还是不算高。现在，ERP产品则发展到比较高的阶段。大多数ERP产品供应商都在模块化的基础上，将软件产品和软件服务进行集成，实现软件产品的技术先进性和个性化设计，为用户提供一体化的解决方案。

同时，先进的信息技术也为ERP提供了技术支持手段，如网络技术、Internet/Intranet技术、条码技术、电子商务技术、数据仓库技术、远程通信技术等，使得各企业在业务往来和数据传递过程中实现电子方式连接；在管理技术上，从内部到外部各环节上，ERP为企业提供了有效的管理工具。由于ERP为企业提供更多更好的功能，帮助企业实现管理信息化和现代化，因而使得企业市场竞争力和综合实力得到提高。

3）管理系统

毫无疑问，管理系统是ERP的基础和依托。一个企业，要根据市场预测制定全面的预算和计划，因此，企业必须实施动态管理。而一个动态的管理模式需要一个运行系统，ERP正是这样一个系统。

ERP是一个集成的信息系统。ERP承诺在建立跨越企业各个部门、各种生产要素和环境的单一应用原则下处理所有的事务，即意味着集成。这种集成应该包括人力资源、财务、销售、制造、任务分派和企业供应链等的各项管理业务。

具体而言，ERP管理系统主要由六大功能目标组成。

一是支持企业整体发展战略经营系统。该系统的目标是在多变的市场环境中建立与企业整体发展战略相适应的战略经营系统，还需要建立与Intranet、因特网相连接的战略系统、决策支持服务体系等。

二是实现全球大市场营销战略与集成化市场营销，也就是实现在预测、市场规模、广告策略、价格策略、服务、分销等各方面进行信息集成和管理集成。

三是完善企业成本管理机制。建立全面成本管理系统，建立和保持企业的成本优势。

四是研究开发管理系统，保证能够迅速地开发适应市场要求的新产品，构筑企业的核心技术体系，保持企业的竞争优势。

五是建立敏捷的后勤管理系统，强调通过动态联盟模式把优势互补的企业联合在一起，用最有效和最经济的方式参加竞争，迅速响应市场瞬息万变的需求。这种敏捷的后勤管理系统能够具有缩短生产准备周期，增加与外部协作单位技术和生产信息及时交互，改进现场管理方法，缩短供应周期等功能。

六是实施准时生产方式，把客户纳入产品开发过程，把销售代理商和供应商、协作单位纳入生产体系，按照客户不断变化的需求同步组织生产，时刻保持产品的高质量、多样性和灵活性。

ERP 对于企业提高管理水平具有重要意义。首先，ERP 为企业提供了先进的信息系统平台。ERP 系统软件不仅功能齐全、集成性强、稳定性好，能够提供准确的信息，而且具备可扩充性。其次，ERP 具有规范的基础管理，促进企业管理水平提高的功能。ERP 实质上就是一套规范的由现代信息技术保证的管理制度。最后，ERP 能够整合企业各种资源，提高资源运作效率。

3．ERP 的功能

ERP 为企业提供的功能是多层面的和全方位的。

一是支持决策的功能。ERP 在 MRPⅡ的基础上扩展了管理范围，给出了新的结构，将企业内部业务流程划分成几个相互协同作业的支持子系统，如财务、市场营销、生产制造、质量控制、服务维护、工程技术等，并在功能上增加了质量控制、运输、分销、售后服务与维护，以及市场开发、人事管理等功能，把企业的制造系统、营销系统、财务系统等都紧密地结合在一起，可以实现全球范围内的多工厂、多地点的跨国经营运作，因而，能够不断地收到来自各个业务过程的运作信息，并且提供了对质量控制、适应变化、客户满意度、效绩等关键问题的实时分析，从而有力地支持企业的各个层面上的决策。

二是为处于不同行业的企业提供有针对性的 IT 解决方案。ERP 已打破了 MRPⅡ局限在传统制造业的格局，把应用扩展到其他行业，如金融业、通信业、零售业等，并逐渐形成了针对于某种行业的解决方案。这一点非常重要，这是因为，不论一个 ERP 软件的功能多么齐全，都无法覆盖所有行业中的特殊需求。一个企业由于其所在行业的原因，既有较为通用的需求，如采购、库存、计划、生产、质检、人事、财务等，还可能有一些与众不同的特殊需求，如石油天然气行业中的勘探与开采、土地使用与租赁，电力行业中的输配电、电表的抄费计价，零售业中的补货、变价、促销等，这些都需要有特殊的功能来解决和管理，从而需要有一套针对该行业的解决方案。为此，有些 ERP 供应商除传统的制造业解决方案外，还推出了商业与零售业、金融业、能源、公共事业、工程与建筑业等行业的解决方案，以财务、人事、后勤等功能为核心，加入每一行业特殊的需求。

三是从企业内部的供应链发展为全行业和跨行业的供应链。当前，任何一个企业都要在全球化的大市场中参与竞争，而竞争的规则就是优胜劣汰，因而，任何一个企业都不可能在所有业务上都成为世界上的佼佼者。如果全部业务都由自己来承担，它必然将面对所有相关领域的竞争对手。因此，只有联合该行业中其他上下游企业，建立一条业务关系紧密、经济利益相连的供应链实现优势互补，才能适应社会化大生产的竞争环境，共同增强市场竞争实力，因此，供应链的概念就由狭义的企业内部业务流程扩展为广义的全行业供

应链及跨行业的供应链。这种供应链或是由物料获取并加工成中间件或成品，再将成品送到消费者手中的一些企业和部门的供应链所构成的网络，或是由市场、加工、组装环节与流通环节建立一个相关业务间的动态企业联盟来进行跨地区、跨行业经营，更有效地向市场提供商品和服务来完成单个企业不能承担的市场功能。这样，ERP 的管理范围亦相应地由企业的内部拓展到整个行业的原材料供应、生产加工、配送环节、流通环节及最终消费者。在整个行业中建立一个环环相扣的供应链，使多个企业能在一个整体的 ERP 管理下实现协作经营和协调运作。把这些企业的分散计划纳入整个供应链的计划中，从而大大增强了该供应链在大市场环境中的整体优势，同时也使每个企业之间均可实现以最小的个别成本和转换成本来获得成本优势。例如，在供应链统一的 ERP 计划下，上下游企业可最大限度地减少库存，使所有上游企业的产品能够准确、及时地到达下游企业，这样既加快了供应链上的物流速度，又减少了各企业的库存量和资金占用。通过这种整体供应链 ERP 管理的优化作用，来到达整个价值链的增值。

这种在整个行业中上下游的管理能够更有效地实现企业之间的供应链管理，以此实现其业务跨行业、跨地区甚至跨国的经营，对大市场的需求做出快速的响应。在它的作用下，供应链上的产品可实现及时生产、及时交付、及时配送、及时交到最终消费者手中，快速实现资本循环和价值链增值，以最大限度地为产品市场提供完整的产品组合，缩短产品生产和流通的周期，使产品生产环节进一步向流通环节靠拢，缩短供给市场与需求市场的距离，既减少了各企业的库存量和资金占用，还可及时地获得最终消费市场的需求信息，使整个供应链均能紧跟市场的变化。通过这种供应链 ERP 管理的优化作用，达到整个价值链的增值。

13.3.3 客户关系管理（CRM）在企业的应用

1．CRM 的概念

当今世界，几乎所有的企业都在宣布坚持“以客户为中心”的理念。但是，怎样把一种好的理念变成企业真实的行动，却不是一件轻而易举的事情。而引进客户关系管理（Customer Relationship Management，CRM）无疑是解决问题的重要举措。CRM 是一种旨在改善企业与客户之间关系的新型管理机制。它通过提供更快速、更周到的优质服务来吸引或保持更多的客户。CRM 集成了信息系统和办公系统等的一整套应用系统，从而确保客户满意度的提高，以及通过对业务流程的全面管理来降低企业的成本。

CRM 在坚持以客户为中心的理念的基础上，重构包括市场营销和客户服务等业务流程。CRM 的目标不仅要使这些业务流程自动化，而且要确保前台应用系统能够改进客户满意度、增加客户忠诚度，以达到使企业获利的最终目标。

需要强调的是脱离后台而只强调前台管理是不够的。只有以客户为中心的应用与能提供客户经验的内部后台系统的集成才可以为整个企业的运作带来所需要的效益。

CRM 实际上是一个概念，也是一种理念；同时，它又不仅是一个概念，也不仅是一种理念，它是企业参与市场竞争的新的管理模式，是一种以客户为中心的业务模型，并由集成了前台和后台业务流程的一系列应用程序来支撑。这些整合的应用系统保证了更令人满意的客户体验，因而会使企业直接受益。

2．CRM 的背景

CRM 的出现体现了两个重要的管理趋势的转变。首先是企业从以产品为中心的模式向

以客户为中心的模式的转变。这种转变有着深刻的时代背景，那就是随着各种现代生产管理和现代生产技术的发展，产品的差别越来越小，产品同质化的趋势则越来越明显，因此，通过产品差异化来细分市场从而创造企业的竞争优势也就变得越来越困难。其次，CRM 的出现还表明了企业管理的视角从“内视型”向“外视型”的转变。众所周知，Internet 及其他各种现代交通、通信工具的出现，使得世界变成了一个地球村，企业与企业之间的竞争，哪怕相隔千里万里，也都变成几乎是面对面的竞争。尤其是在我国，仅仅依靠 ERP 的“内视型”的管理模式已难以适应激烈的竞争，企业必须转换自己的视角，在向“外向型”转变的过程中整合自己的资源。

CRM 听起来是一个很好的概念，然而实施起来却不那么容易。因为 CRM 不只是一套产品，它是触及到企业内许多独立部门的商业理念。

业界分析人士认为，企业的高层管理人员对 CRM 的认识如何至关重要，只有企业管理层接受了 CRM 的理念，CRM 才能在企业里成功地实施，因为只有技术显然是不够的。CRM 需要在整个企业范围内协调关系，开发信息资源。从主导 20 世纪 90 年代的 ERP 系统转变为将注意力集中在客户，通过市场营销和客户服务来优化业务价值的商业模式。在成功实施 CRM 解决方案之前企业需要认同这些新的、不同的商业技巧。企业的商业理念一定要反映在 CRM 应用上，并且在上至公司高层、下到可能与客户发生关系的每位员工之间充分沟通。

3．CRM 的内容

业界一致认为，市场营销和客户服务是 CRM 的支柱性功能。这些是客户与企业联系的主要领域，无论这些联系发生在售前、售中还是售后。

1）客户服务

客户服务是 CRM 的关键内容，是能否形成并保留大量忠诚客户的关键。随着市场竞争的深入，客户对服务的期望值也在不断地提高，已经超出传统的电话呼叫中心的范围。而呼叫中心正在向可以处理各种通信媒介的客户服务中心演变。电话互动必须与 E-mail、传真、网站，以及其他任何客户喜欢使用的方式相互整合。随着越来越多的客户进入互联网通过浏览器来查看他们的订单或提出询问，自助服务的要求发展得也越来越快。

客户服务已经超出传统的帮助平台。“客户关怀”的术语如今用来拓展企业对客户的职责范围。而与客户保持积极主动的关系是客户服务的重要组成部分。客户服务能够处理客户各种类型的询问，包括有关的产品、需要的信息、订单请求、订单执行情况等，还包括高质量的现场服务。

2）市场营销

营销自动化包括商机产生、商机获取和管理，商业活动管理及电话营销等。初步的大众营销活动被用于首次客户接触，接下来是针对具体目标受众的更加集中的商业活动。个性化需求很快成为营销规范，客户的喜好和购买习惯都被列入商家关注的重点。旨在更好地向客户行销带有有关客户特殊需求信息的目录管理和一对一行销应运而生。

市场营销迅速从传统的电话营销转向网站和 E-mail。这些基于 Web 的营销活动给潜在客户更好的体验，使潜在客户以自己的方式在方便的时间查看其需要的信息。销售人员与潜在客户的互动行为并将潜在客户发展为真正客户并保持其忠诚度是使企业盈利的核心因素。

为了获得最大的价值，企业管理层必须与销售人员合作，并对这些商业活动进行跟踪，以激活潜在消费并进行成功/失败研究。市场营销活动的费用管理以及营销事件（如贸易展和研讨会）对未来计划的制定至关重要。

3）共享的客户资料库

共享的客户资料库把市场营销和客户服务连接起来。集成整个企业的客户信息会使企业从部门化的客户联络提高到与客户协调一致的高度。如果一个企业的信息来源相互独立，那么这些信息中必然会存在大量重复、互相冲突的成分。这对企业的整体运作效率将产生负面影响。而动态的、能够被不同部门共享的客户资料库则是企业的一种宝贵资源，同时，它也是 CRM 的基础和依托。

4）分析能力

CRM 的一个重要方面在于它具有使客户价值最大化的分析能力。如今的 CRM 解决方案在提供标准报告的同时，又可提供既定量又定性的及时分析。

深入的智能分析需要统一的客户数据作为切入点，并使所有企业业务应用系统融入到分析环境中，通过对客户数据的全面分析、评估客户带给企业的价值，以及衡量客户的满意度，再将分析结果反馈给管理层，这样便增加了信息分析的价值。企业决策者会权衡这些信息做出更全面、更及时的商业决策。

4．CRM 的解决方案和实施过程

CRM 的根本要求就是与客户建立起一种互相学习的关系，即从与客户的接触中了解他们在使用产品中遇到的问题，以及对产品的意见和建议，并帮助他们加以解决。在与客户互动的过程中，了解他们的姓名、通信地址、个人喜好及购买习惯，并在此基础上进行“一对一”的个性化服务，甚至拓展新的市场需求。比如，客户在订票中心预订了机票之后，CRM 就会根据了解的信息向客户提供唤醒服务或出租车登记等增值服务。因此，可以看到，CRM 解决方案的核心思想就是通过跟客户的“接触”，搜集客户的意见、建议和要求，并通过数据挖掘和分析，提供完善的个性化服务。

一般说来，CRM 由两部分构成，即触发中心和挖掘中心，前者指客户和 CRM 通过电话、传真、Web、E-mail 等多种方式“触发”进行沟通；挖掘中心则是指 CRM 记录交流沟通的信息和进行智能分析。由此可见，一个有效的 CRM 解决方案应该具备如下要素：（1）畅通有效的客户交流渠道（触发中心）。在通信手段极为丰富的今天，能否支持电话、Web、传真、E-mail 等各种触发手段进行交流，无疑是十分关键的。（2）对所获信息进行有效分析（挖掘中心）。（3）CRM 必须能与 ERP 很好地集成。作为企业管理的前台，CRM 的市场营销和客户服务的信息必须能及时传达到后台的财务、生产等部门，这是企业能否有效运营的关键。

CRM 的实现过程具体说来，它包含三方面的工作。一是客户服务与支持，即通过控制服务品质以赢得顾客的忠诚，比如对客户快速准确的技术支持、对客户投诉的快速反应、对客户提供产品查询等。二是客户群维系，即通过与顾客的交流实现新的销售，比如通过交流赢得失去的客户等。三是商机管理，即利用数据库开展销售，比如利用现有客户数据库做新产品推广测试，通过电话促销调查，确定目标客户群等。

5．CRM 的价值

CRM 之所以受欢迎是因为好的客户关系管理对客户和企业都有益。CRM 用户从不断

加强的客户关系管理中明显获益。好的服务不但令人愉快，更能带来巨大价值。带有客户服务的产品的总价值明显高于产品自身。

从另一方面看，企业实施CRM并非出于利他原则，而是认识到客户是其真正的财富。统计显示，68%的客户离开厂家是因为得不到令人满意的客户服务，而企业80%的收入来源于老客户。CRM的成功应用，其效果是显而易见的。

- 较高的满意度，使得企业能够保留老客户，并不断增加新客户。
- 识别利润贡献度最高的客户并给以相应的优厚对待。
- 通过有效目标市场定位，来降低营销成本。
- 引导潜在消费至适当的销售渠道。
- 提供正确的产品来增加销售（交叉销售/纵向销售）。
- 简化部门工作流程来缩短销售周期。
- 通过集中共同活动以减少多余运作。
- 减少由于多个不协调的客户交互点而产生的差错，节省费用。
- 利用客户喜欢的沟通渠道来增加对客户需求的了解。
- 参照与其他客户的联络记录和经验，与目前的客户进行沟通。
- 根据对以前绩效的分析评估未来的销售、营销和客户服务活动。

由于CRM对企业的重大影响，实施CRM项目时需要整个企业范围内的认识与运作。为保持竞争优势，企业必须投资于CRM技术，同时要建立新的业务模型。所有客户信息的集中是成功实施的CRM的核心。CRM这一强有力的企业策略将提高销售、客户忠诚度和企业的竞争优势。

13.3.4 企业门户

随着互联网的快速发展，企业门户已经成为企业优化业务模式、扩展市场渠道、改善客户服务及提升企业形象和凝聚力的强有力手段。企业门户之所以具有极大的吸引力，关键在于它具备广泛的用途和灵活、全面的模型。随着电子商务的发展，企业门户已经成为新型办公环境的重要组成部分。从电子商务应用到企业内部的信息系统，所有用户友好型信息搜集系统都以基于各种技术的企业门户形式出现。不过，如果要给企业门户下一个确切的定义，目前还做不到，因为还没有一个公认的企业门户标准。

1．企业门户的功能

通常，企业需要更高效能且技术统一的平台，以整合当前的网上业务，同时让系统本身能够随时便利升级，以支持未来网上业务的发展。建设集多种功能（如客户关系管理、网上销售、知识管理、内容管理等）于一身的企业门户网站，成为势在必行的上网策略。

一直以来，门户网站仍局限于提供内容、电子邮箱及搜索引擎等基本功能，针对的主要是大众消费类市场；随着互联网应用于企业市场，企业将各类型业务搬到一个开放统一而且安全度很高的网上平台，便成为其电子商务架构中的重要环节。

据相关独立分析员预测，门户网站的趋势将会主导今后几年的企业计算机应用潮流。电子商务需要有更明确的投资回报评估，由此也导致企业对门户网站技术的需求急剧增加。企业门户网站已经显现出提升竞争力的功用：一方面可以让雇员更方便地存取信息，另一方面又可以加强与客户和伙伴之间的联系。

值得一提的是，不同的企业将不尽相同的网络系统连接至单一开放式企业门户网站上，可以大大降低管理成本。因此，企业门户的主要功能有：

- 能够将一个机构现有的互联网址和服务完全合并而且相互兼容。
- 能够支持开放标准和应用编程接口（API），让平台得以轻易容纳新的应用程序。
- 能够接入一个由支持企业门户网站架构的伙伴和专业服务公司所组成的网络。
- 能够多渠道接入网站，如互联网至公司内联网、话音网络、无线网络等。
- 能够以统一的服务作为企业门户网站各种服务的基础，让用户享有多种便利，如一次登入、个人化接口等。当用户进入门户网站的不同部分时，系统可以记住用户的身份以提供合适的信息。

总之，门户网站应该是一个起点，引领用户接触企业最重要的信息、应用和服务。门户网站并非仅为个人计算机用户标准应用而设，它应该能够根据用户的身份、意向、接入方式、接入设备（如移动电话）等设定个性化的信息内容。

2．企业门户的分类

按照实际应用领域，企业门户可以划分为三类：信息门户、知识门户和应用门户。

1）企业信息门户

企业信息门户（Enterprise Information Portal，EIP）的基本作用是为人们提供企业信息，它强调对结构化与非结构化数据的收集、访问、管理和无缝集成。这类门户必须提供数据查询、分析、报告等基本功能，企业员工、合作伙伴、客户、供应商都可以通过企业信息门户非常方便地获取自己所需的信息。

对访问者来说，企业信息门户提供了一个单一的访问入口，所有访问者都可以通过这个入口获得个性化的信息和服务，可以快速了解企业的相关信息。对企业来说，信息门户既是一个展示企业的窗口，也可以无缝地集成企业的业务内容、商务活动、社区等，动态地发布存储在企业内部和外部的各种信息，同时还可以支持网上的虚拟社区，访问者可以相互讨论和交换信息。

在目前企业门户的应用中，信息门户被企业广泛认同。实际上，各企业建立的企业网站都可以算作企业信息门户的雏形。

2）企业知识门户

企业知识门户（Enterprise Knowledge Portal，EKP）是企业员工日常工作所涉及相关主题内容的“总店”。企业员工可以通过它方便地了解当天的最新消息、工作内容、完成这些工作所需的知识等。通过企业知识门户，任何员工都可以实时地与工作团队中的其他成员取得联系，寻找到能够提供帮助的专家或者快速地连接到相关的门户。不难看出，企业知识门户的使用对象是企业员工，它的建立和使用可以大大提高企业范围内的知识共享，并由此提高企业员工的工作效率。

当然，企业知识门户还应该具有信息搜集、整理、提炼的功能，可以对已有的知识进行分类，建立企业知识库并随时更新知识库的内容。目前，一些咨询、服务型企业已经开始建立企业知识门户。

3）企业应用门户

企业应用门户（Enterprise Application Portal，EAP）实际上是对企业业务流程的集成。

它以商业流程和企业应用为核心，把商业流程中功能不同的应用模块通过门户技术集成在一起。从某种意义上说，可以把企业应用门户看成是企业信息系统的集成界面。企业员工和合作伙伴可以通过企业应用门户访问相应的应用系统，实现移动办公、进行网上交易等。

以上 3 类门户虽然能满足不同应用的需求，但随着企业信息系统复杂程度的增加，越来越多的企业需要能够将以上 3 类门户有机地整合在一起的通用型企业门户。按照 IDC 的定义，通用型的企业门户应该随访问者角色的不同，允许其访问企业内部网上的相应应用和信息资源。除此之外，企业门户还要提供先进的搜索功能、内容聚合能力、目录服务、安全性、应用/过程/数据集成、协作支持、知识获取、前后台业务系统集成等多种功能。给企业员工、客户、合作伙伴、供应商提供一个虚拟的工作场所。

3．企业门户的要素

当前，一些企业已经在利用不同的平台和多种互联网/内联网服务开展网上运营。企业门户网站最重要的目标，是将多个系统整合到一个具有可扩充性的平台上，为提供多元化的网上服务做好准备，以最少的投资赚取最高收益。企业可以在基本平台上对各种应用程序加以整合，同时做到支持第三方应用程序所需的标准。

建立互联网服务时应考虑的基本要素如下。

- 战略性思维——评估用户未来的需求，并将这些需要与影响业务发展的因素一并考虑，例如，处理客户数据时个人隐私及安全问题。
- 为用户所需要的不同类型门户网站建立一个门户网站架构。
- 寻找合适的技术供货商——即能够支持各主要标准，并能够将其基本门户网站架构与其他供货商的应用程序整合起来。
- 确定所要建立的门户网站类型，如销售门户网站或知识管理门户网站。制定可量化的目标，并清楚界定投资回报。如果对进展感到满意，就可逐步实行门户网站策略的其他元素。
- 首先小规模地试办项目，确保有一个可行的工作环境。接着，如果用户的工作队伍决定加入新服务，就可相应地扩充项目。

13.3.5　企业应用集成

企业应用集成（Enterprise Application Intergration，EAI）是伴随着企业信息系统的发展而产生和演变的。企业的价值取向是推动应用集成技术发展的原动力，而应用集成的实现反过来也驱动公司竞争优势的提升。EAI 技术是将进程、软件、标准和硬件联合起来，在两个或更多的企业信息系统之间实现无缝集成，使它们就像一个整体一样。EAI 一般表现为对一个商业实体（例如某家公司）的信息系统进行业务应用集成，但当遇到多个企业系统之间进行商务交易时，EAI 也表现为不同公司实体之间的企业系统集成，例如 B2B 的电子商务。

1．EAI 的简要历史

计算机广泛的商业应用开始于 20 世纪 60—70 年代。当时，企业应用的主要目标是利用计算机来代替一部分烦琐的重复性手工工作，以提高生产效率。这时还没有企业数据集成的需求。

到了 20 世纪 80—90 年代，许多企业特别是大型跨国公司在信息系统上投入了巨资，建立了众多的应用信息系统，以帮助企业进行内部或外部业务的处理和管理。由于企业的

传统职能结构，企业整体功能被各个部门所分割，使得信息系统也自然为各个部门所独占，其结果是导致众多关键的信息被封闭在相互独立的系统中，形成一个个所谓的“信息孤岛”。

如何将众多的“信息孤岛”联系起来，以便让不同的系统之间交互信息，EAI 就作为一个企业的需求被提了出来，这时，EAI 的价值和必要性也开始体现。

企业在追求效率和控制成本，或在兼并和收购过程中，对应用集成技术提出了更高的要求，特别是电子商务的兴起。电子商务，这一基于因特网新的商务模式直接导致新的系统集成结构的出现，像 Web 服务技术等。特别是 20 世纪 90 年代，ERP 应用开始流行，也要求新的信息系统能够支持已经存在的应用和数据，这就必须引入 EAI。还有应用供应链管理、Web 应用集成等也对 EAI 起到推动作用。

2．EAI 的内容

EAI 的内容极为广泛，同时，其意义也十分重大，它是企业信息化发展到较高阶段的标志。因为，在企业范围内现有的应用系统和数据库有可能既有几年前的老系统，还可能包括新建系统，需要对它们进行无缝地集成；不同的系统和应用可能包括同样的数据，从而造成了数据冗余、数据的不一致，需要尽量减少数据冗余，并保持所有的数据版本同步更新；企业在激烈的市场竞争中，经常根据需要调整业务流程，必然影响到信息系统的结构和数据，或是建立新的系统等。

总之，EAI 是企业信息系统集成的科学、方法和技术，其目的就是将企业内的应用彼此连接起来，或在企业之间连接起来。

EAI 主要包括两方面：企业内部应用集成和企业间应用集成。EAI 包括的内容很复杂，涉及结构、硬件、软件及流程等企业系统的各个层面。

1）企业内的集成

企业内的应用集成，就是要解决在企业内部业务流程和数据流量，包括业务流程是否进行自动流转或怎样流转，以及业务过程的重要性。对于应用集成，这点非常重要，因为从本质上讲，企业应用集成就是维持数据正确而自动地流转。同时，不同的 EAI 解决方案采取不同的技术途径，而不同的技术途径也就决定了 EAI 处于不同的层次，从应用和技术上综合考虑，EAI 分为界面集成、平台集成、数据集成、应用集成和过程集成。

- 界面集成：这是比较原始和最浅层次的集成，但又是常用的集成。这种方法就是把用户界面作为公共的集成点，把原有零散的系统界面集中在一个新的、通常是浏览器的界面之中。
- 平台集成：这种集成要实现系统基础的集成，使得底层的结构、软件、硬件及异构网络的特殊需求都必须得到集成。平台集成要应用一些过程和工具，以保证这些系统进行快速安全的通信。
- 数据集成：为了完成应用集成和过程集成，必须首先解决数据和数据库的集成问题。在集成之前，必须首先对数据进行标识并编成目录，另外还要确定元数据模型，保证数据在数据库系统中分布和共享。
- 应用集成：这种集成能够为两个应用中的数据和函数提供接近实时的集成。例如，在一些B2B集成中实现CRM系统与企业后端应用和Web的集成，构建能够充分利用多个业务系统资源的电子商务网站。

- 过程集成：当进行过程集成时，企业必须对各种业务信息的交换进行定义、授权和管理，以便改进操作、减少成本、提高响应速度。过程集成包括业务管理、进程模拟等，还包括业务处理中每一步都需要的工具。

2）企业间应用集成

EAI 技术可以适用于大多数要实施电子商务的企业，以及企业之间的应用集成。EAI 使得应用集成架构里的客户和业务伙伴，都可以通过集成供应链内的所有应用和数据库实现信息共享。

传统的 B2B 商务应用了诸如 EDI（电子数据交换）和专用 VAN（增值网络）的技术。然而今天，大多数 B2B 商务则采用了实时性更强的、基于因特网的技术，如基于因特网的消息代理技术、应用服务器，以及像 XML 等新的数据交换标准。

许多公司的供应链系统也可能包括交易系统，新的 EAI 技术可以首先在交易双方之间创建连接，然后再共享数据和业务过程。当然，他们如今不再使用 VAN，而采用因特网作为传输介质。

企业要顺利地开展电子商务，希望其所有的应用之间，以及与其商业伙伴之间都能够实现无缝而及时的通信，这一目标在以前是比较难于实现的，因为 EAI 解决方案比较昂贵，直到新一代支持 EAI 的中间件的出现，才改变了这一面貌。

和 B2B 商务有所不同，B2C 商务需要信息能被更广泛的企业之外的人或客户访问到，所以企业应用要能支持基于 Web 的销售和信息共享。显而易见，B2B 和 B2C 的需要促进了 EAI 技术的发展。

3．集成技术的发展展望

目前市场主流的集成模式有三种，分别是面向信息的集成技术、面向过程的集成技术和面向服务的集成技术。

在数据集成的层面上，信息集成技术仍然是必选的方法。信息集成采用的主要数据处理技术有数据复制、数据聚合和接口集成等。其中，接口集成仍然是一种主流技术。它通过一种集成代理的方式实现集成，即为应用系统创建适配器作为自己的代理。适配器通过其开放或私有接口将信息从应用系统中提取出来，并通过开放接口与外界系统实现信息交互，而假如适配器的结构支持一定的标准，则将极大地简化集成的复杂度，并有助于标准化，这也是面向接口集成方法的主要优势来源。标准化的适配器技术可以使企业从第三方供应商获取适配器，从而使集成技术简单化。

面向过程的集成技术其实是一种过程流集成的思想，它不需要处理用户界面开发、数据库逻辑、事务逻辑等，而只是处理系统之间的过程逻辑，和核心业务逻辑相分离。在结构上，面向过程的集成方法在面向接口的集成方案之上，定义了另外的过程逻辑层；而在该结构的底层，应用服务器、消息中间件提供了支持数据传输和跨过程协调的基础服务。对于提供集成代理、消息中间件及应用服务器的厂商来说，提供用于业务过程集成是对其产品的重要拓展，也是目前应用集成市场的重要需求。

基于 SOA（面向服务架构）和 Web 服务技术的应用集成是业务集成技术上的一次重要的变化，被认为是新一代的应用集成技术。集成的对象是一个个的 Web 服务或者是封装成 Web 服务的业务处理。Web 服务技术由于是基于最广为接受的、开放的技术标准（如 HTTP、SMTP 等），支持服务接口描述和服务处理的分离、服务描述的集中化存储和发布、

服务的自动查找和动态绑定及服务的组合，成为新一代面向服务的应用系统的构建和应用系统集成的基础设施。

13.3.6　供应链管理（SCM）的思想

1．供应链管理的定义

供应链管理（Supply Chain Management，SCM）的核心是供应链。供应链是指一个整体的网络用来传送产品和服务，从原材料开始一直到最终客户（消费者），它凭借一个设计好的信息流、物流和现金流来完成。现代意义的供应链是利用计算机网络技术全面规划供应链中的商流、物流、信息流、资金流等并进行计划、组织、协调和控制。

供应链有两层含义，一层含义是任何一个企业内部都有一条或几条供应链，包括从生产到发货的各个环节；另一层含义是一个企业必定处于市场更长的供应链之中，包括从供应商的供应商到顾客的顾客的每一个环节。供应链是企业赖以生存的商业循环系统，是企业电子商务中最重要的课题。统计数据表明，企业供应链可以耗费企业高达25%的运营成本。

供应链管理是从源头供应商到最终消费者的集成业务流程。它不仅为消费者带来有价值的产品和服务，还为顾客带来有用的信息。供应链管理至少包括以下六大应用功能：需求管理（预测和协作工具）、供应链计划（多工厂计划）、生产计划、生产调度、配送计划、运输计划。新型的供应链管理借助于Internet使这个“供应群”能够实现大规模的协作，成为企业降低成本、提高经营效率的关键。

而在计算机广泛应用之前，企业经常出现因信息传递太慢或错误而误导生产及存货计划的现象。20世纪90年代，一些计算机的制造商（如HP），或生产家庭用品的企业（如宝洁），开始将信息系统做上、下游整合，希望通过正确和快速的信息传递，以及对信息的分析和整合，达到快速反映市场的需求，从而降低库存等目的。因此，有效的供应链管理是建立在高质量的信息传递和共享的基础之上的。

2．供应链与物流

供应链与物流的关系极为密切，而且不可分割。供应链管理是一种管理方法或思想，而物流是在现实经营活动中的物质运动，供应链管理思想是从物流管理的实践中提取出来的，管理的对象是物流；物流分为采购物流、生产物流、销售物流，而供应链管理将这些全部纳入到一个管理体系之中，在供应商、分销商、零销商之间搭建起一个流畅的通道，建立起一个信息共享的机制，从而优化整个供应链，达到降低成本、提高效率等目的。物流的概念诞生在20世纪20年代的美国，当时更多是指商品的移动，如何通过一个载体把商品从生产者手中送到消费者手中。到了20世纪80年代，人们发现以前的概念只是消费物流，忽视了两个环节，即采购环节原材料的物流及在企业内部进行加工生产的生产物流，于是人们又提出来一个整体现代的物流概念。原材料物流对企业来说可能更有意义，因为从采购环节来控制原材料的成本，可以大大降低企业的整体产品的成本，提高产品的竞争力，所以人们这时候发现，通过这种物流的管理，给企业带来的效益非常大，这是物流从狭义到广义的变化。

3．供应链管理是一种管理思想

随着因特网的普及，物流管理很自然地上升为供应链管理。因为在整个交易过程中可能会存在一些矛盾和冲突，供应链管理可以起到弥合整个体系中的矛盾和冲突。例如，以前可能由分销商承担中间运输环节的工作，从供应商处取货送到零售商。后来，零售企业

可能根据自己的效益和规模组建了自己的配送中心，希望分销商能做到：我需要什么货，你必须按照指定时间和地点把货送来。这时分销商自己的物流体系可能就发挥不了太大作用，并且为了按时将零售商需要的商品送到，分销商还需要备好库存，从而加大了成本，这就形成了利益冲突。而通过供应链管理的思想和方法协调供应商、分销商、零售商之间的关系，明确各自在整个体系中所处的角色，搭建一个良好的合作框架。这是各方进一步协同合作的基础。供应链管理一个重要的前提是信息共享，而各种版本 SCM 产品，其核心功能其实是信息传递。如果没有 SCM，也可以依据这样的思想进行人工的信息传递和管理，如派人到超市查看自己产品的库存等，只是这样做的效率比较低。

4．供应链管理的运作模式

供应链中的信息流覆盖了从供应商、制造商到分销商，再到零售商等供应链中的所有环节。其信息流分为需求信息流和供应信息流，这是两个不同流向的信息流。当需求信息（如客户订单、生产计划、采购合同等）从需方向供方流动时，便引发物流。同时供应信息（如入库单、完工报告单、库存记录、可供销售量、提货发运单等）又同物料一起沿着供应链从供方向需方流动。

由于供应链中的企业是一种协作关系和利益共同体，因而供应链中的信息获取渠道众多，对于需求信息来说既有来自顾客也有来自分销商和零售商的；供应信息则来自于各供应商，这些信息通过供应链信息系统而在所有的企业里流动与分享。对于单个企业情况来说，由于没有与上下游企业形成利益共同体，上下游企业也就没有为它提供信息的责任和动力，因此单个企业的信息获取则完全依赖于自己的收集。

处于供应链核心环节的企业要将与自己业务有关（直接和间接）的上下游企业纳入一条环环相扣的供应链中，使多个企业能在一个整体的信息系统管理下实现协作经营和协调运作，把这些企业的分散计划纳入整个供应链的计划中，实现资源和信息共享，增强了该供应链在市场中的整体优势，同时也使每个企业均可实现以最小的个别成本和转换成本来获得成本优势。这种网络化的企业运作模式拆除了企业的围墙，将各个企业独立的信息孤岛连接在一起，通过网络、电子商务把过去分离的业务过程集成起来，覆盖了从供应商到客户的全部过程。对供应链中的企业进行流程再造，建立网络化的企业运作模式是建立企业间的供应链信息共享系统的基石。

统一的信息系统架构是决定信息能否共享的物质技术基础，主要包括为系统功能和结构建立统一的业务标准和建立统一信息交流规范体系等。因为即使某些细节之处没有遵循共同的标准也会影响数据交流和信息共享。例如，供应链中的企业通过 EDI 进行数据交换时，双方必须严格遵守文件的标准格式，任何一方擅自改动格式都将导致对方的系统无法正常工作。

5．供应链管理的技术支持体系

供应链信息系统的建立需要大量信息技术来支持，这是因为供应链管理涉及众多的领域：产品（服务）设计、生产、市场营销（销售）、客户服务、物流供应等。它是以同步化、集成化生产计划为指导，通过采用各种不同信息技术来提高这些领域的运作绩效。

信息技术对供应链的支撑可分为两个层面。

第一个层面是由标识代码技术、自动识别与数据采集技术、电子数据交换技术、互联网技术等基础信息技术构成的。

第二层面是基于信息技术而开发的支持企业生产。

在具体集成和应用这些系统时，不应仅仅将它们视为是一种技术解决方案，而应深刻理解它们所折射的管理思想，涉及的技术和方法主要有：销售时点信息系统（POS），电子自动订货系统（EOS），计算机辅助设计（CAD）和计算机辅助制造（CAM），ERP 和 MRPII，CRM、电子商务等。

13.3.7 商业智能（BI）

商业智能（Business Intelligence）是企业对商业数据的搜集、管理和分析的系统过程，目的是使企业的各级决策者获得知识或洞察力，帮助他们做出对企业更有利的决策。

早在 20 世纪 90 年代末，商业智能技术就被一家计算机权威杂志评选为未来几年最具影响力的信息技术之一。但商业智能技术并不是基础技术或者产品技术，它是数据仓库、联机分析处理 OLAP（Online Analytical Processing）和数据挖掘等相关技术走向商业应用后形成的一种应用技术。

商业智能系统主要实现将原始业务数据转换为企业决策信息的过程。与一般的信息系统不同，它在处理海量数据、数据分析和信息展现等多个方面都具有突出性能。

商业智能系统主要包括数据预处理、建立数据仓库、数据分析及数据展现四个主要阶段。数据预处理是整合企业原始数据的第一步，它包括数据的抽取、转换和装载三个过程。建立数据仓库则是处理海量数据的基础。数据分析是体现系统智能的关键，一般采用联机分析处理和数据挖掘两大技术。联机分析处理不仅进行数据汇总/聚集，同时还提供切片、切块、下钻、上卷和旋转等数据分析功能，用户可以方便地对海量数据进行多维分析。数据挖掘的目标则是挖掘数据背后隐藏的知识，通过关联分析、聚类和分类等方法建立分析模型，预测企业未来发展趋势和将要面临的问题。在海量数据和分析手段增多的情况下，数据展现则主要保障系统分析结果的可视化。一般认为数据仓库、OLAP 和数据挖掘技术是商业智能的三大组成部分。

1．数据仓库：商业智能的基础

对于一个企业来说，最关键也最为重要的是如何以一种有效的方式逐步整理各个业务处理系统中积累下来的历史数据，并通过灵活有效的方式为各级业务人员提供统一的信息视图，从而在整个企业内实现真正的信息共享。数据仓库技术正好满足了这一需求。数据仓库是商业智能系统的基础，如果没有数据仓库，没有企业数据的融合，数据分析就成为无源之水。

数据仓库主要有 4 个重要特征。

- 数据仓库是面向主题的。传统的操作型系统是围绕公司的应用进行组织的。如对一个电信公司来说，应用问题可能是营业受理、专业计费和客户服务等，而主题范围可能是客户、套餐、缴费和欠费等。
- 数据仓库是集成的。数据仓库实现数据由面向应用的操作型环境向面向分析的数据仓库的集成。由于各个应用系统在编码、命名习惯、实际属性、属性度量等方面不一致，当数据进入数据仓库时，要采用某种方法来消除这些不一致性。
- 数据仓库是非易失的。数据仓库的数据通常是一起载入与访问的，在数据仓库环境中并不进行一般意义上的数据更新。

- 数据仓库随时间的变化性。数据仓库中的数据随时间变化的特性表现在三个方面：
  - 数据仓库中的数据时间期限要远远长于操作型系统中的数据时间期限。操作型系统的时间期限一般是60～90天，而数据仓库中数据的时间期限通常是5～10年。
  - 操作型数据库含有“当前值”的数据，这些数据的准确性在访问时是有效的，同样当前值的数据能被更新；而数据仓库中的数据仅仅是一系列某一时刻生成的复杂的快照。
  - 操作型数据的键码结构可能包含也可能不包含时间元素，如年、月、日等；而数据仓库的键码结构总是包含时间元素。

2．OLAP：海量数据分析利器

对于 TB 级的海量数据，联机分析处理 OLAP 无疑是一种有力的数据分析工具。它可以让管理者灵活地对海量数据进行浏览分析。利用多维的概念，OLAP 提供了切片、切块、下钻、上卷和旋转等多维度分析与跨维度分析功能。相对于普通的静态报表，OLAP 更能满足决策者和分析人员对数据仓库数据的分析。

区别于传统的联机事务处理（OLTP）系统，OLAP 有如下 12 条准则。

- OLAP模型必须提供多维概念视图。
- 透明性准则。
- 存取能力推测。
- 稳定的报表能力。
- 客户/服务器体系结构。
- 维的等同性准则。
- 动态的稀疏矩阵处理准则。
- 多用户支持能力准则。
- 非受限的跨维操作。
- 直观的数据操纵。
- 灵活的报表生成。
- 不受限的维与聚集层次。

虽然随着技术的发展，部分准则有所突破，但这些准则仍然是 OLAP 技术的基础。

OLAP 系统架构主要分为基于关系数据库的 ROLAP（Relational OLAP）、基于多维数据库的 MOLAP（Multidimensional OLAP）、基于混合数据组织的 HOLAP（Hybrid OLAP）三种，前两种方式比较常见。ROLAP 表示基于关系数据库的 OLAP 实现。它以关系数据库为核心，以关系型结构进行多维数据的表示和存储。ROLAP 将多维数据库的多维结构划分为两类表：一类是事实表，用来存储数据和维关键字；另一类是维表，即对每个维至少使用一个表来存放维的层次、成员类别等维的描述信息。MOLAP 表示基于多维数据组织的 OLAP 实现。它以多维数据组织方式为核心，使用多维数组存储数据。MOLAP 查询方式采用索引搜索与直接寻址相结合的方式，比 ROLAP 的表索引搜索和表连接方式速度要快得多。

3．数据挖掘：洞察力之源

与展示企业历史和现有信息的静态、动态报表及查询等分析方法不同，数据挖掘是从

数据库中智能地寻找模型，从海量数据中归纳出有用的信息。可以说通过商业智能系统，企业获得洞察力的主要手段就是数据挖掘。

数据挖掘（Data Mining）是从大量的、不完全的、有噪声的、模糊的、随机的数据中，提取隐含在其中的、人们事先不知道的、但又是潜在有用的信息和知识的过程。

数据挖掘技术可分为描述型数据挖掘和预测型数据挖掘两种。描述型数据挖掘包括数据总结、聚类及关联分析等。预测型数据挖掘包括分类、回归及时间序列分析等。

- 数据总结：继承于数据分析中的统计分析。数据总结目的是对数据进行浓缩，给出它的紧凑描述。传统统计方法如求和值、平均值、方差值等都是有效方法。另外，还可以用直方图、饼状图等图形方式表示这些值。广义上讲，多维分析也可以归入这一类。
- 聚类：是把整个数据库分成不同的群组。它的目的是使群与群之间差别很明显，而同一个群之间的数据尽量相似。这种方法通常用于客户细分。在开始细分之前不知道要把用户分成几类，因此通过聚类分析可以找出客户特性相似的群体，如客户消费特性相似或年龄特性相似等。在此基础上可以制定一些针对不同客户群体的营销方案。
- 关联分析：是寻找数据库中值的相关性。两种常用的技术是关联规则和序列模式。关联规则是寻找在同一个事件中出现的不同项的相关性；序列模式与此类似，寻找的是事件之间时间上的相关性，如对股票涨跌的分析等。
- 分类：目的是构造一个分类函数或分类模型（也常称为分类器），该模型能把数据库中的数据项映射到给定类别中的某一个。要构造分类器，需要有一个训练样本数据集作为输入。训练集由一组数据库记录或元组构成，每个元组是一个由有关字段（又称为属性或特征）值组成的特征向量，此外，训练样本还有一个类别标记。一个具体样本的形式可表示为：（v_1，v_2，...，v_n；c），其中v_i表示字段值，c表示类别。
- 回归：是通过具有已知值的变量来预测其他变量的值。一般情况下，回归采用的是线性回归、非线性回归这样的标准统计技术。一般同一个模型既可用于回归，也可用于分类。常见的算法有逻辑回归、决策树、神经网络等。
- 时间序列：是用变量过去的值来预测未来的值。

数据挖掘另一个重要方面是与之相关的方法论。一般的事务处理系统甚至一些只提供报表分析功能的简单商业智能系统，建成以后只需要少量的工程维护工作，而采用数据挖掘技术的商业智能系统往往有很大不同。因为数据挖掘是一个商业理解、数据理解、建模、评估等一系列多次反复、多次调整的过程，并且模型的应用也不是一成不变的，在适当的时候需要更新和重建。所以一般的商业智能项目并不追求一次性工程建设，更倡导的是一种与企业业务紧密联系能够提升企业竞争力的咨询服务，而且熟悉业务和分析方法的分析人员在商业智能系统的应用中起着至关重要的作用。从这一点也能看出为什么说 BI 是企业 MIS 之后更高层次、更具战略意义的应用。

诚然，对于数据挖掘或者商业智能也应有一个客观的认识。从广义上，数据挖掘是在传统数据分析方法基础上，融合了数据库、人工智能等多方面技术形成的知识发现技术。它对企业的信息分析必然产生积极的效果，对企业经营决策的辅助作用也是显而易见的。但是数据挖掘只是一些技术和方法，并非万能，而商业智能系统更多的是为企业提供一个经营分析的环境和一些分析工具。如何切合企业经营实际，从海量的经营数据中挖掘出有

助于企业市场竞争的知识，商业智能系统本身体现并不多。因此，企业洞察力的真正来源是商业智能系统及数据挖掘技术的成功应用和实践。

13.3.8 电子商务

1．什么是电子商务

电子商务英文即 Electronic Commerce，简写为 EC，是指买卖双方利用现代开放的因特网络，按照一定的标准所进行的各类商业活动。主要包括网上购物、企业之间的网上交易和在线电子支付等新型的商业运营模式。产品可以是实体化的，如计算机、汽车、电视；也可以是数字化的，如新闻、影像、软件；也可以直接提供服务，如安排旅游、远程教育等。

电子商务分 3 个方面：即电子商情广告、电子选购和交易，电子交易凭证的交换、电子支付与结算，以及网上售后服务等。

参与电子商务的实体有四类：顾客（个人消费者或集团购买）、商户（包括销售商、制造商、储运商）、银行（包括发卡行、收单行）及认证中心。

狭义的电子商务是指利用 Web 提供的通信手段在网上买卖产品或提供服务；广义的电子商务除上述内容外还包括企业内部的商务活动：如生产、管理、财务等；以及企业间的商务活动：把买家、卖家、厂家和合作伙伴通过因特网、Intranet 和 Extranet 连接起来所开展的业务。从最初的电话、电报、电子邮件，到 20 多年以前开始的电子数据交换 EDI，都可以说是电子商务的雏形；到今天，电子商务已经延伸到商务的各个方面；人们可以通过网络进行原材料查询、采购、产品展示和订购，再到出货、储运及电子支付等一系列完整的贸易过程。从更广泛意义上来说，未来因特网上的活动将是电子商务。

要实现完整的电子商务会涉及很多方面，除买家、卖家外，还要有银行或金融机构、政府机构、认证机构、配送中心等机构的加入才行。由于参与电子商务中的各方在物理上是互不谋面的，因此整个电子商务过程并不是物理世界商务活动的翻版，网上银行、在线电子支付等条件和数据加密、电子签名等技术在电子商务中发挥着重要的不可或缺的作用。

电子商务是网络经济的最重要的组成部分，也是最直接的方式，它的发展对于经济的发展起着至关重要的作用。

2．电子商务的类型

可以对电子商务按参与电子商务交易的对象、电子商务交易的商品内容和进行电子商务的企业所使用的网络类型等对电子商务进行不同的分类。

按参与交易的对象分类，电子商务可以分为如下几类：

（1）企业与消费者之间的电子商务（Business to Customer，B2C）。企业与消费者之间的电子商务是人们最熟悉的一种电子商务类型。网上商店利用 Internet 提供的双向交互通信，完成网上购物的过程。这类电子商务主要是借助于 Internet 所开展的在线式销售活动。最近几年随着 Internet 的发展，这类电子商务的发展异军突起。例如，在 Internet 上目前已出现许多大型超级市场，所出售的产品一应俱全，从食品、饮料到电脑、汽车等，几乎包括所有的消费品。由于这种模式节省了客户和企业双方的时间和空间，大大提高了交易效率，节省了各类不必要的开支，因而这类模式得到了人们的认同，获得了迅速的发展。

（2）企业与企业之间的电子商务（Business to Business，B2B）。两个或是若干各有业

务联系的公司通过 B2B 模式彼此连接起来，形成网上的虚拟企业圈。例如，企业利用计算机网络向它的供应商进行采购，或利用计算机网络进行付款等。B2B 具有很强的实时商务处理能力，使企业能以一种安全、可靠、简便、快捷的方式进行企业间的商务联系活动。

（3）企业与政府间的电子商务（Business to Government，B2G）。B2G 使得政府与企业之间的各项事务都可以涵盖在其中，包括政府采购、税收、商检、管理条例发布等。政府一方面作为消费者，可以通过 Internet 发布自己的采购清单，公开、透明、高效、廉洁地完成所需物品的采购；另一方面，政府对企业宏观调控、指导规范、监督管理的职能通过网络以电子商务方式更能充分、及时地发挥。借助于网络及其他信息技术，政府职能部门能更及时全面地获取所需信息，做出正确决策，做到快速反应，能迅速、直接地将政策法规及调控信息传达于企业，起到管理与服务的作用。在电子商务中，政府还有一个重要作用，就是对电子商务的推动、管理和规范作用。

（4）消费者与消费者之间的电子商务（Customer to Customer，C2C）。C2C 电子商务平台就是通过为买卖双方提供一个在线交易平台，使卖方可以主动提供商品上网拍卖，而买方可以自行选择商品进行竞价。

（5）在线离线/线上到线下（Online To Offline，O2O）。指将线下的商务机会与互联网结合，让互联网成为线下交易的前台，这个概念最早来源于美国。O2O 的概念非常广泛，只要产业链中既可涉及线上，又可涉及线下，就可通称为 O2O。主流商业管理课程均对 O2O 这种新型的商业模式有所介绍及关注。2013 年 O2O 进入高速发展阶段，开始了本地化及移动设备的整合，于是 O2O 商业模式横空出世，成为 O2O 模式的本地化分支。团购就是一种典型的 O2O 模式。

13.4 信息资源管理

什么是信息化？从本质上讲，信息化就是"化"信息。信息与物质、能源、土地等一样，是一种客观存在的资源。一般来说，资源本身并不直接具有价值，资源只有经过开发和利用才具有价值。例如，埋藏在地下的煤矿是一种资源，但不经开发，人们就不能直接利用。因此，煤矿对我们来说不直接具有价值，只有把煤炭从地下开发出来，对我们才具有直接的价值。同样道理，信息是一种资源，只有经过开发，才能具有价值。信息资源的这种开发过程就是"化"信息的过程。一般来说，经过"化"的信息与未经过"化"的信息有着本质的不同。经过"化"的信息，已经成为创造价值的价值，即已经成为企业重要的生产要素。

如何进行信息资源管理，也就是如何"化"信息？詹姆斯·马丁提出了一系列的具有系统性和可操作性的工程化方法，即信息工程方法。

马丁的信息工程方法要解决三个问题，一是要做好战略数据规划，二是要建设好主题数据库，三是围绕主题数据库进行应用开发，而建设好主题数据库则是信息工程方法的重点和关键。

1．做好战略数据规划

马丁在《战略数据规划方法学》一书的前言中指出，"在 20 世纪 70 年代，人们就已看清，对企业和其他组织而言，计算机化的信息乃是具有很高价值的资源。人们还看清了这种信息资源的开发必须有来自最高层的规划，而实施这样的规划又迫切需要一套正规化的，

并且最好是与数据库设计相联系的易于用计算机处理的方法学。”马丁进一步指出，“虽然许多企业早已认识到对信息资源进行规划的必要性，但很少有人知道如何实现这样的规划。某些咨询公司强调了制定这类规划的重要性，但又拿不出什么有效的办法来指导所需信息资源的设计。”按照马丁的观点，一个企业要搞信息化，它的首要任务应该是在企业战略目标的指导下做好企业战略数据规划。一个好的企业战略数据规划应该是企业核心竞争力的重要构成因素，它有非常明显的异质性和专有性，必将成为企业在市场竞争中的制胜法宝。

战略数据规划的工程基础是信息工程方法学。以詹姆斯·马丁为代表的美国学者，总结了信息系统开发的经验与教训，创造性地发现企业数据处理中一个基本规律——数据类和数据之间内在联系是相对稳定的，而对数据处理的业务过程和步骤是经常变化的，明确提出了“信息工程作为一个学科要比软件工程更为广泛，它包括了为建立基于当代数据库系统的计算机化企业所必需的所有相关的学科”（马丁）。而软件和编制程序的学科，实际上是信息工程的一个组成部分。信息工程以前的开发工作，一般都是面向业务过程的。那种面向业务过程的开发方法弊病很大，有一项业务就要开发一个系统，由于数据是业务处理的对象，因而每项业务都不可避免地包含大量的数据和数据处理。

随着系统的增多，就会出现所谓的“数据危机”，系统与系统之间所处理的数据大量地重复、交叉，其后果非常严重，一是使得处理工作量非常浩大，致使系统运行效率低下；二是很容易造成各个系统之间的数据不一致，同一项数据，在不同的系统中取值会不同；三是使得各个系统维护和升级会困难重重；四是各个系统的应用集成会困难异常，甚至是不可能的。

而信息工程把以前开发的顺序倒了过来，由传统的以处理为中心的开发，转变为以数据为中心的开发。其基本思想是：首先，以企业的核心业务和主导业务流程为基础，规划业务数据，着眼于总体数据架构和结构；而后，建立主题数据库；最后，再围绕主题数据库进行积分式的系统开发。信息工程特别强调两条原则，一是高层领导介入的原则，特别是战略数据规划阶段，必须有高层领导介入；二是用户参加开发的原则。

2．建设主题数据库

由于信息工程是以数据为中心的开发思路，因而特别强调信息系统的数据环境建设。马丁把信息系统的数据环境分为四种类型。这四类数据环境反映了由低级到高级的发展过程。

第一类数据环境是数据文件环境。是指早期程序语言，建立的数据存储结构，缺乏数据分析工作。优点是应用开发见效快，缺点是随着应用的增多，冗余的、不一致的数据越来越多，维护与集成十分困难。

第二类数据环境是应用数据库环境。当数据库管理系统出现以后，数据存储结构的建立大大简化了，但是数据分析工作没跟上，用 DBMS 按用户视图“建库”，方便性带来了随意性，于是产生了“数据库风险”。

第三类数据环境是主题数据库环境。经过科学的规划设计与数据分析，用 DBMS 建立具有共享性和一致性的数据库即“主题数据库”，以主题数据库为主的数据环境才是集成化的数据环境，在这种数据环境中才能开发和运行集成化的信息系统。

第四类数据环境是信息检索系统。它是指对一些主题数据库进行萃取和深加工，为企业决策者和管理者提供综合查询和辅助决策准备的数据环境。

在四类数据环境中，主题数据库数据环境占有极为重要的地位，它是企业信息系统开发的重点和中心。

主题数据库，这里的“主题”是指企业的业务主题，例如，一个加工企业的业务主题就是产品的加工，而围绕产品的加工的业务主题有若干业务活动，包括原材料的采购、生产、销售，以及为之服务的产品开发、设计，市场研究，后勤保障等。实质上，主题数据库并不是一个或两个数据库，一般来说，一个有较大规模的企业的主题数据库应当有多个，因此，所谓的主题数据库，其实是一个数据库群。经验表明，一个大型企业的主题数据库个数应在 40 个以内，比如，一家大型银行的主题数据库有 21 个。

主题数据库的突出优点是它具有稳定的结构，不受企业机构或部门变动的影响，不仅能满足本企业管理人员的工作需要，也能为业务伙伴和广告客户提供高效的信息服务。建立主题数据库，要采用一整套信息工程的技术和方法，不过，在集成化信息系统开发初期需要具有一定的规模，但随着系统的扩展，数据库的数目较少增加甚至不增加；而如果不采用主题数据库而采用应用数据库，虽然在开发初期见效快，但随着应用项目的增多，数据库的数目会快速增加。在这种情况下，如果要做到应用项目的信息共享，那么其接口数目会按几何级数增加，以至于达到无法控制的地步。

主题数据库有如下特点。

- 由于一个企业的业务主题具有客观性，这就决定了同行业的不同企业的业务主题的统一性，相应的，其主题数据库的结构也必然是相同的或基本相同的。
- 由于主题数据库不是企业某一部门或某个人的私有数据，它必须纳入企业信息资源的统一管理，因而企业中的不同业务可以共享主题数据库的信息资源。
- 由于主题数据库的信息源具有唯一性，它的数据采集必须是一次性和一地性，并且一次性地进入系统，因而避免了数据的不一致。
- 主题数据库的结构具有稳定性、原子性、演绎性和规范性，因而，便于系统开发的自动化，以及便于系统维护、升级和集成。

3．基于主题数据库的应用开发

当在战略数据规划的指导下，主题数据库开发完成以后，企业及其各个部门或机构就可以根据本部门的需要，围绕主题数据库来开发业务处理系统。应当指出，围绕主题数据库的信息系统开发一般来说是高效的，开发出的系统也可以是健壮的。

第 14 章

信息系统基础知识

信息系统是指处理、加工信息的系统，在信息化的大环境中，我们把很多传统的人工处理的信息系统向计算机化的信息系统转化。在此过程中，要求掌握信息系统的基本概念，以及信息系统建设的相关知识。

14.1 信息系统

信息系统（Information System，IS）一般泛指收集、存储、处理和传播各种信息的具有完整功能的集合体。在这里，信息系统并没有强调收集、存储、处理和传播信息所用的工具。作为一般意义上的信息系统，在任何时代、任何社会都会存在，然而，只有到了今天，信息系统的概念才被创造出来，并得到相当程度的普及，这是因为，在当今社会，信息系统总是与计算机技术和互联网技术的应用联系在一起，因此，现代的信息系统总是指以计算机为信息处理工具，以网络为信息传输手段的信息系统。因此，现今只要说到信息系统，一般来说，就是指的这样的信息系统，而不必特意说明是“现代”信息系统。

现代信息系统与 50 年来计算机技术和网络技术的发展保持同步。随着社会的进步和技术的发展，信息系统的内容和形式也都在不断发生着巨大的变化。与其他事物一样，信息系统也经历了一个从低级到高级、从局部到全局、从简单到复杂的发展过程。信息系统大致经历了四个发展阶段。

第一阶段：电子数据处理阶段

计算机应用于企业是从简单数据处理开始的。计算机发明以后的一段时期，计算机仅仅用于科学计算。后来，计算机程序设计人员将计算机应用领域进行了拓展，开始尝试用计算机进行数据处理，从而开辟了计算机更广阔的应用领域。不过，最早的计算机在数据处理中的应用，仅着眼于减轻人们在计算方面的劳动强度，如用于计算工资、统计账目等，属于电子数据处理（EDP）业务，对企业单项业务进行处理，较少涉及管理内容。

第二阶段：事务处理阶段

随着企业业务需求的增长和技术条件的发展，人们逐步将计算机应用于企业局部业务的管理，如财会管理、销售管理、物资管理、生产管理等，即计算机应用发展到对企业的局部事务的管理，形成了所谓的事务处理系统，即 TPS（Transaction Process System），但它并未形成对企业全局的、整体的管理。

第三阶段：管理信息系统阶段

人们常说的信息系统大多指支持各部门和机构管理决策的信息系统，因此，信息系统一般又称为“管理信息系统”（Management Information System，MIS）。管理信息系统一词最早出现在 20 世纪 80 年代初，此后，在应用中得到了快速的发展。人们从不同的角度对它进行了定义，比较被广泛认可的定义是：“管理信息系统是用系统思想建立起来的，以电子计算机为基本信息处理手段，以现代通信设备为基本传输工具，且能为管理决策提供信息服务的人机系统。即管理信息系统是一个由人和计算机等组成的，能进行管理信息的收集、传输、存储、加工、维护和使用的系统。”

在 MIS 阶段，信息系统形成了对企业全局性的、整体性的计算机应用。密斯强调以企业管理系统为背景，以基层业务系统为基础，强调企业各业务系统间的信息联系，以完成企业总体任务为目标，它能提供企业各级领导从事管理需要的信息，但其收集信息的范围还更多地侧重于企业内部。

第四阶段：决策支持系统阶段

当前，计算机信息系统已经从管理信息系统发展成更强调支持企业高层决策的决策支持系统（DSS），即决策支持系统阶段。

因特网技术的发展和应用，在很大程度上拓展和提升了信息系统的功能和作用，其最大的特点是通过因特网将众多的孤立的信息系统（即“信息孤岛”）联系起来，形成在更大程度上实现信息共享的大范围的基于网络互联的信息系统。因特网技术应用于企业内部信息系统，可以促进企业内综合 MIS、DSS 功能，并以办公自动化技术为支撑的办公信息系统的实施。企业信息系统的目标为：借助于自动化和互联网技术，综合企业的经营、管理、决策和服务于一体，以求达到企业和系统的效率、效能和效益的统一，使计算机和互联网技术在企业管理和服务中能发挥更显著的作用。

这里需要指出的是，信息系统的四个发展阶段，它们之间的关系并不是取代关系，而是互相促进、共同发展的关系，也就是说，在一个企业里，以上四个阶段的信息系统，可能同时存在，也可能只有其中的一种、两种或三种。更高级的是几种信息系统互相融合成一体，比如，ERP、SRM 等就是这种情况。

14.1.1 信息系统的功能

信息系统的功能就是信息系统的使用价值，也就是信息系统所能做的事情和所起的作用。信息系统的功能是一个“多面体”，即从不同的角度分析，其功能是不一样的。

1．需求功能和实现功能

从企业业务需求的角度看，信息系统的功能可以分为需求功能和实现功能。所谓需求功能，是企业的业务对信息系统提出的要求，比如，一个生产零售产品的企业，它非常需要如该产品的市场容量、竞争对手的情况等信息。而实现功能则是指，对于业务需求信息，由于各种客观条件的限制，信息系统只能提供其中的一部分，这部分就是实现功能。

2．初级功能和高级功能

从发展阶段的角度看，可分为初级功能和高级功能，而且是初级功能逐步地发展为高级功能。在这方面，有很多人都做了非常深入的研究，具有代表性的是著名的“诺兰模型”。信息技术应用于组织中，一般都要经历从初级到高级，从不成熟到不断成熟的成长阶段。诺兰（Nolan）第一次总结了这一规律，1973 年提出了信息系统发展的阶段理论，被称为

诺兰阶段模型。到 1980 年，诺兰进一步完善该模型，把信息技术的成长过程划分为六个阶段。

在 Nolan 模型中，第一阶段即“初装（Initiation）阶段”。它的标志是组织安装了第一台计算机并引入了自动化概念，同时初步开发了管理应用程序。在该阶段，计算机的作用被初步认识。一般大多发生在财务部门。

第二阶段即“蔓延（Contagion）阶段”。其标志是随着自动化的扩展（从少数部门扩散到多数部门，并开发了大量的应用程序）而导致的计算机系统的急增。在该阶段，数据处理能力发展得最为迅速，但同时也出现了许多亟待解决的问题，如数据冗余性、不一致性、难以共享等。

第三阶段即“控制（Control）阶段”。其标志是试图遏制快速上升的计算机服务成本并将数据处理置于控制之下。为了加强组织协调，出现了由企业领导和职能部门负责人参加的领导小组，对整个企业的系统建设进行统筹规划，特别是利用数据库技术解决数据共享问题。

第四阶段即“集成（Integration）阶段”。其标志是各种各样的系统和技术集成为内在统一的系统，数据处理发展进入再生和控制发展时期。

第五阶段即“数据管理（Data Administration）阶段”。其标志是完全集成的、基于数据的系统发展和实施的结束。

第六阶段即“成熟（Maturity）阶段”。其标志是公司数据管理的日益成熟，可以满足单位中各管理层次的要求，从而实现信息资源的管理。

随后，Nolan 又将该模型的六个阶段划分为两个时代，即计算机时代和信息时代，其中，前三个阶段构成计算机时代，后三个阶段进入信息时代。Nolan 模型能够帮助一个组织识别其所处的阶段从而确立相应的发展战略，是信息化战略管理的重要理论工具。

Nolan 的两个时代划分理论于 20 世纪 90 年代之后逐渐不能适应信息技术发展的需要，为此，Nolan 又提出了一种理解组织内部信息技术进化的新框架，该框架将信息技术的发展分为三个阶段，即数据处理（DP）阶段、信息技术（IT）阶段和网络（Network）阶段。

DP 阶段（20 世纪 60—80 年代）：信息技术主要在一个组织的操作层面和管理层面起作用，其主要功能是使一些专门的工作自动化，如支持各种指令处理的事物处理系统（Transactions Processing Systems，TPS）、提供资源配置和控制信息的 MIS 系统就是在该阶段发展起来的。

IT 阶段（20 世纪 80—90 年代中期）：信息技术在一个组织中的发展进入战略管理层面，强调知识工作者对信息技术的利用，如财务分析员、证券经纪人和生产规划者等常用 PC 工作站来分析“what if”（如果……怎样）之类的问题。

Network 阶段（20 世纪 90 年代中期之后）：信息技术不再能够单方面地使组织取得他们所寻求的业务效果，信息技术与组织人员及其工作整合为一种网络化的组织形式以创造 10 倍速的生产率。Nolan 的三个时代划分更符合 20 世纪 90 年代之后信息技术发展的新趋势，因而也更有利于战略信息管理者理解信息技术并制定相应的发展战略。

Nolan 模型提出以后不久，美国学者 Edgar Schein 也提出了一种称为“新的信息技术发展阶段模型”的理论，其特点是将信息技术进化过程与组织变迁过程联系起来考察，有助

于形成一种整体化的认识，该模型包括四个阶段。

第一阶段，投资或启动阶段。组织决定在新的信息技术方面投资，新技术能够带来明显的益处，则将顺利进入第二阶段，如果该阶段没有用户参与或发生了供应商方面的问题，那么就会延迟信息技术的进化，并导致成本超出预算、项目缺乏管理及其他不可预期的技术问题。

第二阶段，技术学习和适应阶段。用户通过学习如何利用技术来完成任务，如果用户有机会更好地理解新技术及其益处，则顺利进入第三阶段，如果过早地控制技术发展，那么就会影响用户的学习过程，并导致缺乏进一步开发信息技术潜力的动机等问题。

第三阶段，管理控制阶段。组织认识到信息技术的重要性并对系统发展和实施过程予以精确的控制，如果控制过程能够确保各种应用的成本—效益的成功，则顺利进入第四阶段，如果出现过多的控制，就会导致创新热情的丧失、新技术扩散的失败乃至从头再来等问题。

第四阶段，大范围的技术转移阶段。新的信息技术如局域网技术等将转移到组织的其他部门，信息技术知识也将随技术向用户转移，信息技术成为组织结构的有机组成部分。

Schein 的“信息技术阶段论”提供了 IT 角色的一种新视角，其主要作用仍在于引导战略信息管理者识别组织所处的信息技术发展阶段，并结合所采用的信息技术类型制定针对性的发展战略和方案，它同时也表明，不适当的、过早的和过多的控制不利于信息技术的应用和普及，并容易导致各种停滞问题。

如何认识诺兰模型，在我国一直存在着争论。张亚明提出了一些自己的见解，值得关注。张亚明认为，诺兰的模型有需要进一步理解和值得商榷的地方。实际上，第一、二阶段带有很大的自发性和盲从性，单纯以提高组织事物处理的效率为主，表现为许多自动化“孤岛”；第三、四、五阶段为有高层领导参与的自觉管理阶段，其内部的基于局域网的数据管理逐渐达到成熟，对战略起支持性作用；Nolan 模型的第六阶段（成熟阶段）划分太“粗”，实质应划分成两个阶段，即第六阶段的“信息资源管理”和第七阶段的“成熟”。其中第六阶段将信息不仅视为资源，而且视为战略资源来管理，为组织创造战略机会服务。此外这阶段管理的重点是组织的“外部信息”和企业的“知识管理”。这个时期，企业的内联网（Intranet）、外联网（Extranet）逐渐完善并发展成熟。第七阶段的“成熟”实质是基于 Internet 架构上战略的体现，是基于内联网（Intranet）、外联网（Extranet）的再一次彻底的集成和整合，表现为信息技术和战略融为一体，企业驾驭信息技术的能力达到真正的成熟，企业真正成为完全的“数字化企业”。

3．通用功能和专业功能

任何一个信息系统都必须具备一些基本的、通用功能。同时，作为一个信息系统，都必然是在特定环境下产生的，而且为了实现特定的目标，因此，每个信息系统一般还要具有一些特有的、专业的功能。

一般来说，一个信息系统应当具有的通用功能是多方面的，而且是复杂的，但是，以下基本功能是任何一个信息系统所必不可少的。

1）数据库功能

任何一个信息系统，都应当以数据库为基础，因此，数据库功能是一个信息系统的基本功能之一，而数据库的功能之一就是提供统一格式、统一结构的数据，从而使各种统计工作大大

简化，降低信息成本。

2）*存储信息功能*

存储信息是信息系统的主要功能之一。一个没有信息系统的企业尽管也存储大量信息，比如，一般企业里都建有档案系统，保存了大量信息，但是许多企业运行过程中产生的信息，却很少保存，而信息系统则可以保存更多的过程信息。据了解，有的大型企业一个信息系统就存储近 10 GB 的数据，这些数据如果不是用信息系统存储，是不可能实现的。

3）*检索信息的功能*

利用信息系统可以及时地提供满足不同要求的信息，以满足使用者的需要。许多传统手工方式查阅档案的工作，在信息系统中，都变成了在键盘上的几下操作。

4）*信息分析功能*

在信息系统中，不但可以非常方便地进行信息的存储和检索，更重要的是可以对信息进行分析，从而为决策提供支持意见。

詹姆斯·马丁在《大转变》一书中指出，“企业的知识存在于日益复杂的软件之中。”世界连锁之王——沃尔玛的许多“绝招”就存在于它的信息系统之中。现代企业真正有效力的竞争武器，往往就体现在它的信息系统。

4．整体功能和局部功能

建设信息系统是企业信息化非常重要和关键的内容。因此，信息系统必须能够体现企业的总体战略，也就是说，企业信息系统必须为企业总体战略服务，这就是信息系统的整体功能。同时，一个信息系统必然由若干个部分组成，即可以分为多个子系统，而每一个子系统都要有自己的功能，这种功能就是局部功能。

为了使信息系统的整体功能和每个部分的局部功能都能得到加强，有必要在整体规划的指导下，建立起全企业信息系统功能模型，然后，再根据各子系统和程序模块的具体情况，对各个局部功能进行优化、整合，从而形成在市场竞争中具有快速反映能力的、完善的信息系统。

14.1.2　信息系统的类型

当今的信息系统，由于其广泛的应用，已经发展成为一个极为庞大的家族，而且几乎每个信息系统的内部构成都非常复杂。为了充分认识信息系统，有必要对其进行分类。但是，如何进行分类，并不是一个简单的问题。目前对于信息系统有很多的分类方法：如从计算机应用的角度，可以分成人工信息系统和基于计算机的信息系统；从独立性的角度，可以分成独立信息系统和综合信息系统；从处理方式的角度，可以分成批处理信息系统和联机处理信息系统等。下面介绍两种重要的信息系统分类方法。

1．以数据环境分类

目前对于信息系统最为权威的分类方法是世界信息系统大师詹姆斯·马丁的分类。马丁从信息系统的数据环境的角度出发，对信息系统进行分类。

马丁在著作《信息工程》和《战略数据规划方法学》中将信息系统的数据环境分为四种类型，并认为清楚地了解它们之间的区别是很重要的，因为它们对不同的管理层次，包括高层管理的作用是不同的。

第一类数据环境：数据文件。其特征是：没有使用数据库管理系统，根据大多数的应

用需要，由系统分析师和程序员分散地设计各种数据文件。其特点是简单，相对容易实现。但随着应用程序增加，数据文件数目剧增，导致很高的维护费用；任何应用上的微小变化都将引起连锁反应，使修改和维护工作既缓慢费用又高昂，并很难进行。

第二类数据环境：应用数据库。这类信息系统，虽然使用了数据库管理系统，但没达到第三类数据环境那种共享程度。分散的数据库为分散的应用而设计，实现起来比第三类数据环境简单。像第一类数据环境一样，随着应用的扩充，应用数据库的个数，以及每个数据库中的数据量也在急剧增加，随之而导致维护费用大幅度增高，有时甚至高于第一类数据环境。该类数据环境还没有发挥使用数据库的主要优越性。

第三类数据环境：主题数据库（Subject Data Bases）。主题数据库信息系统所建立的一些数据库与一些具体的应用有很大的独立性，数据经过设计，其存储的结构与使用它的处理过程都是独立的。各种面向业务主题的数据，如顾客数据、产品数据或人事数据，通过一些共享数据库被联系和体现出来。这种主题数据库的特点是：经过严格的数据分析，建立应用模型，虽然设计开发需要花费较长的时间，但其后的维护费用很低。最终（但不是立即）会使应用开发加快，并能使用户直接与这些数据库交互使用数据。主题数据库的开发需要改变传统的系统分析方法和数据处理的管理方法。但是，如果管理不善，也会蜕变成第二类或第一类数据环境。

第四类数据环境：信息检索系统（Information Retrieval Systems）。一些数据库被组织得能保证信息检索和快速查询的需要，而不是大量的事务管理。软件设计中要采用转换文件、倒排表或辅关键字查询技术。新的字段可随时动态地加入到数据结构中。有良好的最终用户查询和报告生成软件工具。大多数用户掌握的系统都采用第四类数据库。这种环境的特点是：比传统的数据库有更大的灵活性和动态可变性。一般应该与第三类数据环境共存，支持综合信息服务和决策系统。

在数据库技术逐渐普及，软件工程方法得到推广的一二十年中，不同的企业单位开展计算机应用，形成了多种多样的数据环境；这些企业的高层领导和数据处理部门或迟或早都会认识到，需要对现存的数据环境进行改造，以保证信息需求的不断提高，克服现行计算机在数据处理方面的问题，提高科学管理水平，这就需要进行战略数据规划。还有一些企业单位，计算机应用刚刚起步，或者准备开展计算机应用，需要吸取别人的经验教训，避免走错路、走弯路。如果有先进的方法论作为指导，就会快速、科学地实现目标，这就更需要这种战略性的、奠基性的规划工作——战略数据规划。对于前一类单位，通过战略数据规划，尽快地将现有数据环境转变到第三类、第四类数据环境，以保证高效率高质量地利用数据资源。对于后一类单位，战略数据规划是整个计算机应用发展规划的基础与核心，是计算机设备购置规划、人才培训规划和应用项目开发规划的基础。两类单位搞战略数据规划的共同目标是分析、组织、建立企业稳定的数据结构，规划各种主题数据库的实施步骤和分布策略，为企业管理计算机化打下坚实的基础。

2．以应用层次分类

一个公司的管理活动可以分成四级：战略级、战术级、操作级和事务级，相应的，信息系统就其功能和作用来看，也可以分为四种类型，即战略级信息系统、战术级信息系统、操作级信息系统和事务级信息系统。不同级别的信息系统的所有者和使用者都是不同的。一般来说，战略级的信息系统的所有者和使用者都是企业的最高管理层，对于现代公司制企业，就是企业的董事会和经理班子；战术级信息系统的使用者一般是企业的中层经理及

其管理的部门；操作级信息系统的使用者一般是服务型企业的业务部门，例如，保险企业的保单处理部门；事务级信息系统的使用者一般是企业的管理业务人员，例如，企业的会计、劳资员等。

14.1.3 信息系统的发展

信息系统经过 20 多年的发展，目前已经得到极为广泛的应用，已经成为社会信息化的支柱和主导。特别是近 10 年来，由于因特网的普及使得信息系统的开发环境得到了根本性的改善，在应用的深度和广度上都有了空前的发展。同时，信息系统的开发和设计工具和方法层出不穷，使得信息系统在有用性和方便性上都有很大的提高。目前，信息系统已经发展成为一个庞大的家族，已有的类型在不断完善和提高，更有一些新的类型在不断创造出来。

下面介绍信息系统几方面新的进展。

1．基于因特网的信息系统

自 20 世纪 90 年代初互联网出现以来，在世界范围内得到了飞速发展，为信息资源开发和信息系统建设提供了一个非常广阔的平台。特别是 1996 年以后，内联网（Intranet）和外联网（Extranet）的出现，诞生了一种新的信息系统——基于互联网的信息系统。

基于互联网的信息系统除具有一般信息系统的特性以外，它还有许多特殊的性质。

1）内联网

内联网是企业内部的计算机网络，但它使用了因特网的一些标准通信协议及图形化的 Web 浏览器来支持企业内部的计算机应用，提供部门内部及部门之间的直至全公司范围内的信息共享。内联网与因特网有很多不同。

- 内联网只局限在企业内部，它用防火墙将自己封闭起来。因此它的范围要比因特网小得多。
- 由于内联网是企业的内部网络，因而它不需要经过公共的通信线路，从而便于管理，也不需要另外付费。
- 内联网的运行效率要比因特网高得多。
- 内联网不会产生因特网那样的安全问题。

2）外联网

在当今世界，许多企业，特别是那些大型的跨国企业，总部与其下属单位或分支机构可能相距甚远，有些大型跨国公司的分支机构可能分布在世界上的几十个国家和地区。但同时，这样的企业也是一个企业，也是一个独立的市场主体，它要以一个独立的法人身份参与市场竞争。它们怎样才能做到这点呢？企业的外联网就比较好地解决了问题。

外联网是内联网概念和系统的进一步扩展，它借助于因特网把企业的联网范围扩大到远离企业本部的组织和部门，以及与企业关系密切的单位，以使得企业与合作伙伴之间可以通过计算机网络共享信息资源。也可以通过外联网实现电子商务。

可以看出，外联网不仅适合于在分布于不同地理位置的企业集团内部共享信息资源，而且适用于在企业与其供销链伙伴之间交换信息，同时还适合于企业的驻外部门与企业之间的数据通信。

外联网与内联网相比，有许多优越的地方。

内联网用防火墙把系统限制在企业内部，从而保证了企业信息系统的安全。但是，在使用防火墙时，虽然安全性得到提高，但同时也限制了防火墙外的用户、潜在的顾客和合作伙伴访问企业公开的和有一定密级的信息资源的自由，甚至将集团公司位于外地的子公司也挡在防火墙之外，这种情况显然对企业不利。而外联网就较好地解决了这类问题。

基于因特网的信息系统就是建立在因特网、内联网和外联网之上的信息系统。这样的信息系统真正实现了数据的跨地理空间的分布处理，同时，也真正能够把信息系统建成具有实时操作功能的信息系统。

2．多媒体信息系统

传统的信息系统的数据类型比较单一，一般最多的是文本数据，有时也夹杂着一些图形和声音等类型的数据，其原因是数据类型的多少完全取决于计算机的处理能力。所谓"多媒体技术"其实质是计算机能够处理的数据类型的集合。事实上，这个集合的元素是随着计算机技术的发展而不断增加的。随着多媒体技术的发展，计算机能处理的数据类型也随之增多，除文本、图形、声音等类型外，计算机还可以处理如图像、动画、影像等不同的类型，而且同一种类型又可分为不同的子类。例如同是图像类，又可分为彩色和黑白两种子类型，其实，彩色和黑白还可以细分。

多媒体信息系统建设中经常用到的一项重要技术就是虚拟现实技术。虚拟现实是近年来出现的高新技术，它综合集成了计算机图形学、人机交互技术、传感与测量技术、仿真、人工智能、微电子等科学技术。虚拟现实技术通过系统生成虚拟环境，用户通过计算机进入虚拟的三维环境，可以运用视觉、听觉、嗅觉、触觉感官与人体的自然技能感受逼真的虚拟环境，身临其境地与虚拟世界进行交互作用，乃至操纵虚拟环境中的对象，完成用户需要的各种虚拟过程。虚拟现实技术主要应用于工程设计、数据可视化、飞行模拟、模拟实验、多媒体远程教育、远程医疗、旅游娱乐等方面。

多媒体信息系统是信息技术，特别是多媒体技术发展发展的产物，它符合信息技术的发展趋势和人们日益增长的需要。由于多媒体技术的日益成熟，使得计算机、通信及多媒体技术逐步趋向融合，构成以互联网为基础的信息基础设施。

多媒体信息系统的基本特点是：以人为中心的计算，即以符合人的习惯的方式进行信息交互，因此需要基于多功能感知的智能接口，甚至提供一个人性化的应用环境；它支持多用户的以多种媒体进行的实时交互；它是一个集成的系统，能完成任务的全过程；在信息检索方面，由于传统检索方法中所用的关键字检索不能代表多媒体信息中的丰富信息，因此多媒体信息系统的检索是基于内容的信息检索。

多媒体信息系统的应用是极为广泛的，包括远程教育、远程医疗、数字图书馆、协同设计、并行工程、协同指挥系统等。数字图书馆、远程教育、虚拟企业和分布式协同设计与制造等对人们的生活和工作将产生深远影响。

网络通信与虚拟现实技术的结合具有诱人的前景和巨大的潜在应用价值，它将在某种程度上改变人类的思维方式和时空观，实现真正意义上的远程交互式教育，对于我国这样人口众多，教育普及程度低的国家的教育事业发展无疑具有重大、深远的现实意义。基于Internet的远程教育，是为了迎合信息社会的到来对教育提出的全新的需求和挑战。

数字图书馆被认为是21世纪信息产业主要的发展方向，它的目标不是简单地把图书等资料数字化并放到网上，而是要进行以人为中心的计算，使读者能方便地在浩瀚的数据中

找到所需的信息，并进行交流。

3．海量信息系统

我们知道，任何一个信息系统都必须有数据，但是，不同的系统所拥有的数据的多少经常是大不相同的。平常的信息系统所拥有的数据量一般是 KB 级、MB 级，多的也就是 GB（1G＝1024M）级，然而，随着计算机和互联网应用的深入，出现了一些数据量非常大的信息系统，这就是海量信息系统。海量信息系统的数据量一般都是 TB（1T＝1024G）级、PB（1P＝1024T）级，甚至更多。

海量信息系统是信息系统领域的一个重要发展方向，而且应用也非常广泛，一般用在比较专业的领域。比如，我们常说的数字地球、数字城市、数字图书馆等；还比如，卫星遥感信息系统，据说美国的卫星遥感数据就有几万张光盘，以及油田的物探数据等都属于海量信息系统。

哲学辩证法有一条原理，就是量的积累可以引起质的变化。这条原理也适用于信息系统的开发。海量信息系统由于其数据量非常大，因而与普通信息系统有非常不同的性质，而且对软/硬件条件的要求也与普通信息系统大不一样。比如，数字地球由分布式大型数据库构成，由于处理的是海量数据，因而需要具有相应的高密度、高速率、大规模（海量）空间数据存储、压缩、处理技术，对信息提取和分析技术的智能化程度也有更高要求，这些都是对现有计算机软/硬件设计、技术的有力挑战。

4．智能信息系统

人们常说的人工智能其实有两方面的含义，一是作为学科的人工智能，它是指“研究机器智能和智能机器的高新技术学科，是模拟、延伸和扩展新一代计算机的前沿阵地，是探索人脑奥秘的重要科学途径和计算机应用的广阔领域”（涂序彦，1995）；二是作为工程技术的人工智能，它是指一系列应用人工智能原理的技术和方法，例如，专家系统技术、知识库技术、模式识别、机器学习技术、机器推理技术、机器人技术等。

智能信息系统是人工智能与信息系统的结合体，这里的人工智能是作为工程技术的人工智能。

由低级向高级发展是一条普遍的规律，信息系统的发展也正是遵循着这一规律。在计算机出现以前，信息系统事实上已经存在，只不过那时人们并没有自觉地认识到这点。那时的信息系统都是针对特定用户群的信息需求而设计和建立起来的人工系统，它也能够进行信息的采集、组织、存储、检索、分析综合与传递。但是，效率比较低，人们为获取和处理信息所花费的成本也比较大。随着现代系统思想、方法和技术的不断完善，信息系统从传统的纯手工系统发展到基于计算机技术的普通信息系统。随着人工智能研究和应用的深入，人工智能的技术、方法和思想也逐步被引入信息系统的开发过程，于是，信息系统就从普通信息系统发展到智能信息系统。

智能信息系统的基本特征集成了人和机器两方面的优点，而避免了两方面的缺点，因而是信息系统发展的高级阶段。

一般来说，人的优点是智能水平高，富有创造性、灵活性和主动性，能够进行模糊记忆；缺点是人易受主观感情和客观干扰的影响，易产生疲劳，容易遗忘，且记忆随时间的延长而减弱。计算机的优点是能够对大量的信息进行高速、精确的处理、存储和管理；缺点是智能水平低，缺少创造性、主动性和灵活性。这里应当指出，普通信息系统由于应用

了计算机技术和网络技术，因而，已经具备了大部分人和机器的优点，而避免了其缺点。而智能信息系统则是在更高级的程度上运用了人工智能技术，从而更多地具备了人和机器智能的优点，更多地避免了其缺点。智能信息系统至少在以下方面比普通信息系统得到提高。

一是做到人—机协调性。为了实现人—机协调性，智能信息系统要进行人—机的合理分工，以便实现人—机的智能结合。另一方面，在人—机智能接口设计中，尽量采用模式识别、自然语言理解方面的技术和多媒体技术，实现人—机友好交互，特别是人—机自然语言对话。

二是做到人—机智能的结合。智能信息系统最大的优势就是人和机器智能的有机结合，即在人—机合理分工的基础上，提高信息系统的智能水平。例如，使系统具有知识表达、知识存储、知识管理、知识获取和知识利用的能力。

三是实现智能信息系统的辅助决策和决策的功能。诺贝尔经济学奖得主、世界人工智能大师赫伯特·西蒙有一句名言："管理就是决策。"毫无疑问，决策是人的智能的重要组成部分，它也应是智能信息系统的主要功能之一。而专家系统是人工智能中出现最早，也是发展最成熟的子学科，因而，一般来说，专家系统已成为智能信息系统的具有支柱意义的技术。

5．其他方面

近年来，由于计算机技术、因特网技术及其他相关技术的迅猛发展，具有更多技术含量的新产品不断涌现，使得信息系统呈现多元的发展趋势。上述四种信息系统只是几个典型例子，其他新型信息系统还很多，例如，基于数据仓库的信息系统、计算机集成制造系统（CIMS）等。

14.2 信息系统建设

本节主要介绍信息系统建设。

14.2.1 信息系统建设的复杂性

大型信息系统的建设是资金密集、技术密集的宏大而复杂的系统工程，它的复杂性不仅来自于计算机、网络和通信等一系列现代技术方面的因素，更重要的是来自于系统建设和管理体制方面的关系和联系，还来自于企业之外的社会因素。因此，信息系统的建设与侧重于技术的系统工程，如大型电站、大型工厂等有很大区别，通常一项技术工程通过规划设计、研制生产、安装调试后即可进入稳定的运行。而信息系统则不然，其复杂性既由于技术的复杂，又由于管理的复杂，而且当两者结合起来以后，其复杂性就尤其突出。其系统需求和内/外部条件总是处于不断的变化之中，因此，没有一个信息系统建好后是一劳永逸的，系统的修改、维护、升级、扩展，甚至重建都会经常发生。

因此，全面而深入地认识信息系统的复杂性对于系统开发非常重要。

1．信息系统开发的复杂性

企业信息系统是一个公认的复杂系统，其复杂性既由于技术的复杂，又由于管理的复杂，而且当两者结合起来以后，其复杂性就尤其突出。

技术复杂性是多方面的，一是信息系统涉及的技术跨越多个领域：有计算机科学与技

术领域，包括软件、硬件等；通信领域，包括有线通信、无线通信等；网络领域，包括局域网、广域网、内联网、外联网、互联网等；以及数学等。因此，在信息系统的开发中，如何选择技术，以及选择什么样的技术，无疑都有很多难题。二是信息系统所用到的技术都是当今的热门技术，其发展变化异常迅速，很多技术的生命周期很短，有些技术的生命周期为一两年，也有的技术刚刚出生不久，就面临被更新的技术所替代的命运。面对层出不穷的新技术，信息系统的开发者必须进行判断。三是与新技术相适应的新产品也层出不穷，令人目不暇接。对此，信息系统的开发者要做出准确的选择。

管理方面的复杂性更为突出。在开始开发的初期，一般来说，很难给出企业信息系统一个明确的轮廓，究竟要建成一个什么样的信息系统对管理人员来说还都是一个谜，即使开发者设计出来一个完整的方案，也很难向领导及业务人员解释清楚，说服所有的人。至于信息系统将会带来什么效益，更难明确地回答。

2．信息系统运行的复杂性

一个信息系统的开发的复杂性是很大的，但是，运行的困难性可能会更大，这是因为，信息系统的运行需要有科学的管理体制、良好的管理基础、完善的管理机构、合理的管理流程，还要有管理人员，尤其是领导的支持和参与。而要做到这些，就要首先解决管理上的问题，例如管理体制和管理机构的调整、业务流程的优化或重组，管理人员习惯观念等改变。

一个牵涉到企业全局的信息系统要做到良好的运行，需要特别解决好如下四个问题。

一是要解决基础数据的问题。一个信息系统所处理的对象主要是数据，因此，数据的质量问题十分重要。软件工程中有一句话："输入的是垃圾，输出的肯定也是垃圾"。这就是说，信息系统不可能"化腐朽为神奇"，不可能把垃圾数据处理成有用的数据。而一些信息系统的需求单位，恰恰是基础数据不全、不准或不一致。所谓数据不全是指只有部分信息系统所需要的数据，例如，一个企业有 10 个下属单位，只有 6 个下属单位有数据，其他则没有，这样一来，该系统的运行效果就必然大打折扣。所谓数据不准，就是指一些基础数据有差错，由此，必然影响系统的可靠性。所谓数据不一致，是指同一项数据在不同的地方取不同的值。

二是领导介入的问题。企业的信息系统绝不仅仅是一个软件的使用，它要涉及企业的组织流程，涉及企业的机构调整，涉及因信息系统的运行而使企业发生许多新的变化，这些都决定了信息系统不是一个技术的问题；同时，许多问题和障碍也不是仅靠技术人员就能解决的。信息系统的运行需要企业最高领导层的介入，而一些企业的管理层对此却缺乏足够的认识。在一些企业里，最高管理层把信息系统的建设和运行交给信息技术部门就算万事大吉，有这样做法的企业，其信息系统的良好运行将成为问题。

三是最终用户问题。企业信息系统的最终用户，也就是信息系统的使用者往往是那些企业管理机构的业务人员。信息系统运行的难题是要让这些业务人员接受信息系统，首先，需要改变他们长时间形成的一些工作习惯，这往往比较困难。再者，这些业务人员需要熟悉并掌握信息系统的一些技术和工作方法，这也需要一个比较复杂的过程。

四是系统分析师。信息系统是复杂的人—机工程，因而最需要的人才是既懂经营管理又懂计算机技术的专家型的人才，也就是系统分析师。而很多企业在建设和运行信息系统时，恰恰缺少的就是系统分析师。

3．信息系统维护改造的复杂性

由于企业内外部环境和企业经营管理需求的不断变化，信息系统的维护改造是不可避免的，特别是随着计算机设备的不断降价，个人计算机越来越多地出现在管理人员的办公桌上，要发挥这些设备的效益，必须把它们互联起来，既要满足每个管理人员的信息需要，又要给高层领导提供及时的决策信息。这时，人们才惊讶地发现，分散的开发所带来的严重后果：修改原先的软件，重新组织数据，连成一个统一的大系统，所耗费的人力和资金比重新建立还要多；甚至采取维护和修改的办法是根本行不通的。美国 20 世纪 80 年代初的统计表明，国防部每年支付的软件维护费为 20 亿美元，估计到 20 世纪 80 年代末要高达 160 亿美元；80 年代初美国全国每年软件维护费耗资 200 亿美元。系统维护问题就像病魔似的缠住了数据处理的发展，这就是人们所说的"数据处理危机"。传统的数据处理开发方法所遭到的一些失败，也是这种危机的表现。例如，IBM 公司为日本的两家报社开发自动化系统，总编辑在终端上如何工作的问题一直搞不清楚，使 IBM 公司损失 200 万美元；而这些无畏的开发者不懈努力，在几年后使美国的新闻管理工作自动化，设计文档资料竟达 2 400 页。这使人们开始怀疑，从需求分析开始的传统的生命周期开发方法论，是否符合大型复杂信息系统的开发？

以詹姆斯·马丁（James Martin）为代表的美国学者，总结了这一时期数据处理发展的正反两方面经验，在有关数据模型理论和数据实体分析方法的基础上，再加上他发现的企业数据处理中的一个基本原理——数据类和数据之间的内在联系相对稳定，而对数据的处理过程和步骤则是经常变化的，于 1981 年出版了《信息工程》一书，提出了信息工程的概念、原理和方法，勾画了一幅建造大型复杂信息系统所需要的一整套方法和工具的宏伟图景。第二年出版了《总体数据规划方法论》一书，对信息工程的基础理论和奠基性工作——总体数据规划方法，从理论上到具体做法上详加阐述。经过几年的实践和深入研究，詹姆斯·马丁于 20 世纪 80 年代中期又出版了《信息系统宣言》一书，对信息工程的理论与方法加以补充和发展，特别是关于自动化的自动化思想，关于最终用户与信息中心的关系，以及用户在应用开发中应处于恰当位置的思想，都有充分的发挥；同时加强了关于原型法（Prototyping）、第四代语言和应用开发工具的论述；最后，向与信息工程有关的各类人员，从企业领导到程序员，从计算机制造商到软件公司，以"宣言"（Manifesto）式的忠告，提出了转变思维和工作内容的建议，实际上这是一系列关于建设高效率、高质量的复杂信息系统的经验总结。至此，可以认为信息工程作为一个学科已经形成了，用信息工程方法作为指导，成功地开发了越来越多的信息系统，逐渐引起了人们的注意。

14.2.2　信息系统的生命周期

信息系统与其他事物一样，也要经历产生、发展、成熟和消亡的过程。我们把信息系统从产生到消亡的整个过程称为信息系统的生命周期。

一般来说，信息系统的生命周期分为四个阶段，即产生阶段、开发阶段、运行阶段和消亡阶段。

1．信息系统的产生阶段

信息系统的产生阶段，也是信息系统的概念阶段或者是信息系统的需求分析阶段。这一阶段又分为两个过程，一是概念的产生过程，即根据企业经营管理的需要，提出建设信息系统的初步想法；二是需求分析过程，即对企业信息系统的需求进行深入地调研和分析，并形成需求分析报告。

2．信息系统的开发阶段

信息系统的开发阶段是信息系统生命周期中最重要和最关键的阶段。该阶段又可分为五个阶段，即总体规划、系统分析、系统设计、系统实施和系统验收阶段。

1）总体规划阶段

信息系统总体规划是系统开发的起始阶段，它的基础是需求分析。以计算机和互联网为工具的信息系统是企业管理系统的重要组成部分，是实现企业总体目标的重要工具。因此，它必须服从和服务于企业的总体目标和企业的管理决策活动。总体规划的作用主要有：

- 指明信息系统在企业经营战略中的作用和地位。
- 指导信息系统的开发。
- 优化配置和利用各种资源，包括内部资源和外部资源。
- 通过规划过程规范企业的业务流程。

一个比较完整的总体规划，应当包括信息系统的开发目标、信息系统的总体架构、信息系统的组织结构和管理流程、信息系统的实施计划和信息系统的技术规范等。

2）系统分析阶段

系统分析阶段的目标是为系统设计阶段提供系统的逻辑模型。

系统分析阶段以企业的业务流程分析为基础，规划即将建设的信息系统的基本架构，它是企业的管理流程和信息流程的交汇点。

系统分析的内容主要应包括组织结构及功能分析、业务流程分析、数据和数据流程分析、系统初步方案等。

3）系统设计阶段

系统设计阶段是根据系统分析的结果，设计出信息系统的实施方案。系统设计的主要内容包括系统架构设计、数据库设计、处理流程设计、功能模块设计、安全控制方案设计、系统组织和队伍设计、系统管理流程设计等。

4）系统实施阶段

系统实施阶段是将设计阶段的结果在计算机和网络上具体实现，也就是将设计文本变成能在计算机上运行的软件系统。由于系统实施阶段是对以前的全部工作的检验，因此，在系统实施阶段用户的参与特别重要。如果在系统设计阶段以前，用户处于辅助地位，而到了系统实施阶段以后，用户就应逐步变为系统的主导地位。

5）系统验收阶段

信息系统实施阶段结束以后，系统就要进入试运行。通过试运行，系统性能的优劣、是否做到了用户友好等问题都会暴露在用户面前，这时就进入了系统验收阶段。

3．信息系统运行阶段

当信息系统通过验收，正式移交给用户以后，系统就进入了运行阶段。一般来说，一个性能良好的系统，运行过程中会较少出现故障，即使出现故障，也较容易排除；而那些性能较差的系统，运行过程中会故障不断，而且可能会出现致命性故障，有时故障会导致系统瘫痪。因此，长时间的运行是检验系统质量的试金石。

另外，要保障信息系统正常运行，一项不可缺少的工作就是系统维护。在软件工程中，

把维护分为四种类型，即排错性维护、适应性维护、完善性维护和预防性维护。一般在系统运行初期，排错性维护和适应性维护比较多，而到后来，完善性维护和预防性维护就会比较多。

4．信息系统消亡阶段

通常人们比较重视信息系统的开发阶段，轻视信息系统的运行阶段，而几乎完全忽视信息系统的消亡阶段。其实，这样做是片面的。因为计算机技术和互联网技术的发展十分快速，新的技术、新的产品不断出现；同时，由于企业处在瞬息万变的市场竞争的环境之中，在这种情况下，企业开发好一个信息系统，并希望它能一劳永逸地运行下去，是不现实的。企业的信息系统经常不可避免地会遇到系统更新改造、功能扩展，甚至报废重建的情况。对此企业应当在信息系统建设的初期就要注意系统的消亡条件和时机，以及由此而花费的成本。

14.2.3 信息系统建设的原则

为了能够适应开发的需要，在信息系统规划设计及系统开发的过程中，必须要遵守一系列原则，这是系统成功的必要条件。下面几条原则是信息系统开发时常用的原则。

1．高层管理人员介入原则

一个信息系统其建设的目标总是为企业的总体目标服务的，否则，这个系统就不应当建设。而真正能够理解企业总体目标的人必然是那些企业高层管理人员，只有他们才能知道企业究竟需要什么样的信息系统，而不需要什么样的信息系统，也只有他们才知道企业有多大的投入是值得的，而超过了这个界限就是浪费。这点是那些身处某一部门的管理人员，或者技术人员所无法做到的。因此，信息系统从概念到运行都必须有企业高层管理人员介入。当然，这里的“介入”有着其特定的含义，它可以是直接参加，也可以是决策或指导，还可以是在政治、经济、人事等方面的支持。

这里需要说明的是，高层管理人员介入原则在现阶段已经逐步具体化，那就是企业的“首席信息官”（Chief Information Officer，CIO）的出现。CIO 是企业设置的相当于副总裁的一个高级职位。负责公司信息化的工作，主持制定公司信息规划、政策、标准，并对全公司的信息资源进行管理控制的公司行政官员。在大多数企业里，CIO 是公司最高管理层中的核心成员之一。毫无疑问，深度介入信息系统开发建设，以及运行是 CIO 的职责所在。

2．用户参与开发原则

在我国，流行着信息系统开发中所谓“用户第一”或“用户至上”的原则。当然，这个原则并没有错，一个成功的信息系统，必须把用户放在第一位，这应该毫无疑义。但是，究竟应当怎么“放”？怎么“放”才算是第一位？都没有一个确切的标准。而马丁提出的“用户参与开发原则”就把“用户第一原则”具体化了。

用户参与开发原则主要包括如下几项含义。

一是“用户”有确定的范围。究竟谁是用户？我们通常把“用户”仅仅理解成用户单位的领导，其实这是很片面的。当然，用户单位领导应该包括在用户范围之内，但是，更重要的用户或核心用户，是那些信息系统的使用者，而用户单位的领导只不过是辅助用户或外围用户。

二是用户，特别是那些核心用户，不应只参与某一阶段的开发，而应当参与全过程的

开发，即用户应当参与从信息系统概念规划和设计阶段，直到系统运行的整个过程。而当信息系统交接以后，他们就成为系统的使用者。

三是用户应当深度参与系统开发。用户以什么身份参与开发是一个很重要的问题。一般说来，参与开发的用户人员，既要以甲方代表身份出现，又应成为真正的系统开发人员，与其他开发人员融为一体。

3．自顶向下规划原则

在信息系统开发的过程中，经常会出现信息不一致的问题，这种现象的存在对于信息系统来说往往是致命的，有时一个信息系统会因此而遭到报废的结果。研究表明，信息的不一致是由计算机应用的历史性演变所造成的，它通常发生在没有一个总体规划的指导就来设计实现一个信息系统的情况之下。因此，坚持自顶向下规划原则对于信息系统的开发和建设来说至关重要。自顶向下规划的一个主要目标是达到信息的一致性。同时，自顶向下规划原则还有另外一个方面，那就是这种规划绝不能取代信息系统的详细设计。必须鼓励信息系统各子系统的设计者在总体规划的指导下，进行有创造性的设计。

4．工程化原则

在 20 世纪 70 年代，出现了世界范围内的“软件危机”。所谓软件危机是指一个软件编制好以后，谁也无法保证它能够正确地运行，也就是软件的可靠性成了问题。软件危机曾一度引起人们，特别是工业界的恐慌。经过探索，人们认识到，之所以会出现软件危机，最主要的原因，软件产品是一种个体劳动产品，最多也就是作坊式的产品。因此，没有工程化是软件危机发生的根本原因。此后，发展成了“软件工程”这门工程学科，在一定程度上解决了软件危机。

信息系统也经历了与软件开发大致相同的经历。在信息系统发展的初期，人们也像软件开发初期一样，只要做出来就行，根本不管实现的过程。这时的信息系统，大都成了少数开发者的“专利”，系统可维护性、可扩展性都非常差。后来，信息工程、系统工程等工程化方法被引入到信息系统开发过程之中，才使得问题得到了一定程度的解决。

其实，工程化不仅是一种有效的方法，它也应当是信息系统开发的一项重要原则。

5．其他原则

对于信息系统开发，人们还从不同的角度提出了一系列原则，例如：

- 创新性原则，用来体现信息系统的先进性。
- 整体性原则，用来体现信息系统的完整性。
- 发展性原则，用来体现信息系统的超前性。
- 经济性原则，用来体现信息系统的实用性。

14.2.4　信息系统开发方法

企业信息系统对于企业信息化的重要意义不言而喻。从实际运行的效果来看，有些信息系统运行得很成功，取得了巨大的经济效益和社会效益；但也有些信息系统效果并不显著，甚至有个别信息系统开始时还能正常运行，可时间一长，系统就故障不断，最后走上报废之路。是什么导致这样截然不同的结果呢？当然，这里的原因可能很复杂，但有一个原因是十分重要和关键的，那就是信息系统的开发方法问题。

我们知道，信息系统是一个极为复杂的人—机系统，它不仅包含计算机技术、通信技

术，以及其他的工程技术，而且它还是一个复杂的管理系统，还需要管理理论和方法的支持。下面简单介绍几种最常用的信息系统开发方法。

1．结构化方法

结构化方法是由结构化系统分析和设计组成的一种信息系统开发方法。在本书前面的几章中，较详细地介绍了该方法，因此，这里只做简单的介绍。读者如果想了解结构化生命周期法的详细内容，可阅读本书的有关章节。

结构化方法是目前最成熟、应用最广泛的信息系统开发方法之一。它假定被开发的系统是一个结构化的系统，因而，其基本思想是将系统的生命周期划分为系统调查、系统分析、系统设计、系统实施、系统维护等阶段。这种方法遵循系统工程原理，按照事先设计好的程序和步骤，使用一定的开发工具，完成规定的文档，在结构化和模块化的基础上进行信息系统的开发工作。结构化方法的开发过程一般是先把系统功能视为一个大的模块，再根据系统分析设计的要求对其进行进一步的模块分解或组合。

结构化生命周期法主要特点介绍如下。

- 开发目标清晰化。结构化方法的系统开发遵循“用户第一”的原则，开发中要保持与用户的沟通，取得与用户的共识，这使得信息系统的开发建立在可靠的基础之上。
- 工作阶段程式化。结构化方法每个阶段的工作内容明确，注重开发过程的控制。每一阶段工作完成后，要根据阶段工作目标和要求进行审查，这使阶段工作有条不紊，也避免为以后的工作留下隐患。
- 开发文档规范化。结构化方法每一阶段工作完成后，要按照要求完成相应的文档，以保证各个工作阶段的衔接与系统维护工作的便利。
- 设计方法结构化。结构化方法采用自上而下的结构化、模块化分析与设计方法，使各个子系统间相对独立，便于系统的分析、设计、实现与维护。结构化方法被广泛地应用于不同行业信息系统的开发中，特别适合于那些业务工作比较成熟、定型的系统，如银行、电信、商品零售等行业。

2．快速原型法

快速原型法是一种根据用户需求，利用系统开发工具，快速地建立一个系统模型展示给用户，在此基础上与用户交流，最终实现用户需求的信息系统快速开发的方法。在现实生活中，一个大型工程项目建设之前制作的沙盘，以及大型建筑的模型等都与快速原型法有同样的功效。应用快速原型法开发过程包括系统需求分析、系统初步设计、系统调试、系统检测等阶段。用户仅需在系统分析与系统初步设计阶段完成对应用系统的简单描述，开发者在获取一组基本需求定义后，利用开发工具生成应用系统原型，快速建立一个目标应用系统的最初版本，并把它提交给用户试用、评价，根据用户提出的意见和建议进行修改和补充，从而形成新的版本，再返回给用户。通过这样多次反复，使得系统不断地细化和扩充，直到生成一个用户满意的方案为止。

快速原型法具有开发周期短、见效快、与业务人员交流方便的优点，特别适用于那些用户需求模糊，结构性比较差的信息系统的开发。

3．企业系统规划方法

企业系统规划方法（Business System Planning，BSP）是最早由 IBM 公司于 20 世纪 70 年代研制并使用的一种企业信息系统开发的方法。虽然 30 多年的时间过去了，但是，这种

方法对于今天我国企业信息系统建设仍然具有一定的指导意义。

BSP 方法是企业战略数据规划方法和信息工程方法的基础和，也就是说，后两种方法是在 BSP 方法的基础上发展起来的，因此，了解并掌握 BSP 方法对于全面掌握信息系统开发方法是有帮助的。BSP 方法的目标是提供一个信息系统规划，用以支持企业短期的和长期的信息需求。

4．战略数据规划方法

詹姆斯·马丁是世界级的信息系统大师，他提出的战略数据规划方法是信息系统开发极为重要的一种方法。《战略数据规划方法学》是马丁阐述该方法的一本专著，本书只对战略数据规划方法简单介绍。

对于战略数据规划方法，《战略数据规划方法学》的前言中指出，“在 20 世纪 70 年代，人们就已看清，对企业和其他组织而言，计算机化的信息乃是具有很高价值的资源。人们还看清了，这种信息资源的开发必须有来自最高层的规划，而实施这样的规划又迫切需要一套正规化的，并且最好是与数据库设计相联系的易于用计算机处理的方法学。”马丁进一步指出，“虽然许多企业早已认识到对信息资源进行规划的必要性，但很少有人知道如何实现这样的规划。某些咨询公司强调了制定这类规划的重要性，但又拿不出什么有效的办法来指导所需信息资源的设计。”按照马丁的观点，一个企业要建设信息系统，它没有必要急着去购置设备，也没有必要马上组织软件开发和上网，它的首要任务应该是在企业战略目标的指导下做好企业战略数据规划。一个好的企业战略数据规划应该是企业核心竞争力的重要构成因素，它有非常明显的异质性和专有性，必将成为企业在市场竞争中的制胜法宝。

战略数据规划方法的要点主要有：

- 数据环境对于信息系统至关重要。企业数据环境是随着企业的发展不断变化的，也是企业发展的基础条件。信息系统建设极大影响着企业的未来发展方向，对企业的数据环境提出了更高的要求。把静态的、独立的信息资源通过战略数据规划重建企业数据环境，使其成为集成化、网络化的信息资源，对一个现代化企业来说是更为迫切的任务。
- 四种数据环境。在信息系统发展的历程中共有四类数据环境，即数据文件、应用数据库、主题数据库和信息检索系统。
- 建设主题数据库是信息系统开发的中心任务。这里的主题数据库并不是指数据库的大小，也不是指数据库的功能是什么，而是指哪些数据库是面向企业的业务主题的，哪些不是面向业务主题的。所谓业务主题，就是指企业的核心业务和主导流程。比如，对于一个机加工企业来说，生产机件产品就是其核心业务，相应的，围绕核心业务建立的数据库就是企业的主题数据库。而对于一个保险企业来说，围绕着保单处理的数据库就是企业的主题数据库。
- 围绕主题数据库搞好应用软件开发。

5．信息工程方法

信息工程方法是詹姆斯·马丁创立的面向企业信息系统建设的方法和实践。信息工程方法与企业系统规划方法和战略数据规划方法是一种交叉关系，即信息工程方法是其他两种方法的总结和提升，而其他两种方法则是信息工程方法的基础和核心。

信息工程是计算机信息系统发展到比较成熟阶段的产物，它不仅为大型信息系统的开

发给出了方法和技术，而更重要的是它在理论与实践的结合上对大型信息系统的开发提出了相应的开发策略和原则，而这些策略和原则对于信息系统的成功开发和应用都是至关重要的。虽然，信息工程是在 20 世纪 80 年代末期发展起来的，但是，在今天，仍然对信息系统的开发具有重要的指导价值。

信息工程方法与信息系统开发的其他方法相比，有一点很大的不同，就是信息工程不仅是一种方法，它还是一门工程学科。它第一次把信息系统开发过程工程化了。所谓工程化，就是指有一整套成熟的、规范的工程方法、技术、标准、程序和规范，使得开发工作摆脱随意性和多变性，其目标是信息系统的开发走上智能化、程序化和自动化的道路。

6．面向对象方法

面向对象方法是对客观世界的一种看法，它把客观世界从概念上看成一个由相互配合而协作的对象所组成的系统。信息系统开发的面向对象方法的兴起是信息系统发展的必然趋势。数据处理包括数据与处理两部分。但在信息系统的发展过程的初期却是有时偏重这一面，有时偏重那一面。在 20 世纪 70—80 年代，偏重数据处理者认识到初期的数据处理工作是计算机相对复杂而数据相对简单。因此，先有结构化程序设计的发展，随后产生面向功能分解的结构化设计与结构化分析。偏重于数据方面人员同时提出了面向数据结构的分析与设计。到了 20 世纪 80 年代，兴起了信息工程方法，使信息系统开发发展到了新的阶段。

信息工程在实际应用中既表现出其优越性的一面，同时也暴露了一些缺点，比如，过于偏重数据，致使应用开发受到影响。而面向对象方法则集成了以前各种方法的优点，避免了各自的一些缺点。

面向对象的分析方法是利用面向对象的信息建模概念，如实体、关系、属性等，同时运用封装、继承、多态等机制来构造模拟现实系统的方法。传统的结构化设计方法的基本点是面向过程，系统被分解成若干个过程。而面向对象的方法是采用构造模型的观点，在系统的开发过程中，各个步骤的共同的目标是建造一个问题域的模型。在面向对象的设计中，初始元素是对象，然后将具有共同特征的对象归纳成类，组织类之间的等级关系，构造类库。在应用时，在类库中选择相应的类。

第 15 章

标准化知识

根据考试大纲，本章要求考生掌握如下知识点：

- 标准化意识，标准化组织机构，标准的内容、分类、代号与编号规定，标准制订过程。
- 国际标准、国家标准、行业标准、企业标准。
- 代码标准、文件格式标准、安全标准、互联网相关标准、软件开发规范和文档标准、基于构件的软件标准。

15.1 标准化概述

标准化是一门综合性学科，其工作内容极为广泛，可渗透到各个领域。标准化工作的特征包括横向综合性、政策性和统一性。

1．什么是标准

为在一定的范围内获得最佳秩序，对活动或其结果规定共同的和重复使用的规则、导则或特性的文件，称为标准。该文件经协商一致制定并经一个公认机构的批准。标准应以科学、技术和经验的综合成果为基础，以促进最佳社会效益为目的。

2．什么是标准化

为在一定的范围内获得最佳秩序，对实际的或潜在的问题制定共同的和重复使用的规则的活动，称为标准化。它包括制定、发布及实施标准的过程。标准化的重要意义是改进产品、过程和服务的适用性，防止贸易壁垒，促进技术合作。

3．标准化的实质和目的是什么

“通过制定、发布和实施标准，达到统一”是标准化的实质。“获得最佳秩序和社会效益”则是标准化的目的。

4．标准化的对象是什么

在国民经济的各个领域中，凡具有多次重复使用和需要制定标准的具体产品，以及各种定额、规划、要求、方法、概念等，都可称为标准化对象。

标准化对象一般可分为两大类：一类是标准化的具体对象，即需要制定标准的具体事物；另一类是标准化总体对象，即各种具体对象的总和所构成的整体，通过它可以研究各种具体对象的共同属性、本质和普遍规律。

5．制定标准要经过哪几个阶段

一项标准的出台一般要经过六个阶段：第一阶段，申请阶段；第二阶段，预备阶段；第三阶段，委员会阶段；第四阶段，审查阶段；第五阶段，批准阶段；第六阶段，发布阶段。

若在开始阶段得到的文件比较成熟，则可省略其中的一些阶段。

15.2 标准的层次

根据制定机构和适用范围的不同，标准可分为若干个级别，如国际标准、国家标准、行业标准和企业标准等。

1．国际标准

国际标准是指由国际联合机构制定和公布，提供各国参考的标准。目前，世界上有许多个国际和区域性组织在制定标准或技术规则，其中最大的是国际标准化组织（International Standards Organization，ISO）、国际电工委员会（IEC）和国际电信联盟（ITU）。ISO、IEC、ITU 标准均为国际标准。此外，被 ISO 认可、收入 KWIC 索引中的其他 25 个国际组织制定的标准，也视为国际标准。

2．国家标准

国家标准是指由政府或国家级的机构制定或批准，适用于全国范围的标准，例如：

- GB。中华人民共和国国家标准，由国家质量监督检验检疫总局批准，国家标准化管理委员会公布。 GB/T为推荐性国家标准，GB/Z为指导性国家标准，GSB为国家实物标准。
- ANSI（American National Standards Institute）。美国国家标准协会标准。
- BS（British Standard）。英国国家标准。
- JIS（Japanese Industrial Standard）。日本工业标准。

3．行业标准

行业标准是指由行业机构、学术团体或国防机构制定，并适用于某个业务领域的标准，例如：

- IEEE（Institute of Electrical and Electronics Engineers）。美国电气和电子工程师学会标准。
- GJB。中华人民共和国国家军用标准，由国防科学技术工业委员会批准，适合于国防部门和军队。
- DOD-STD（Department Of Defense STanDards）。美国国防部标准，适用于美国国防部门。
- MIL-S（MILitary Standards）：美国军用标准，适用于美国军队内部。
- 国内地方标准代号一般以DB开头。

4．企业标准

企业标准是指一些大型企业或机构，由于工作需要制定的适用于本企业或机构的标准。国内企业标准代号一般以 Q 开头。

15.3 软件开发规范和文档标准

软件开发过程是一种工程行为，是多种人员之间的协作过程，必须遵循一定的规范和标准，包括软件生存期各阶段的标准和文档写作标准。

1．软件开发规范标准

1988 年，前国家标准局批准并发布了《GB 8566—1988 计算机软件开发规范》，将软件生命周期划分为可行性研究与计划、需求分析、概要设计、详细设计、实现、组装测试、确认测试、使用和维护 8 个阶段。

1990 年，前国家技术监督局同时批准并发布了《GB/T 12504—1990 计算机软件质量保证计划规范》和《GB/T 12505—1990 计算机软件配置管理计划规范》。GB/T 12504 规定了在制定软件质量保证计划时应该遵循的统一的基本要求，GB/T 12504 则规定了在制定软件配置管理计划时应该遵循的统一的基本要求。

1995 年，GB8566—1988 的替代标准，《GB/T 8566—1995 信息技术 软件生存期过程》使用软件生命周期的 7 个过程取代了原来的 8 个阶段，这 7 个过程是管理过程、获取过程、供应过程、开发过程、运行过程、维护过程和支持过程。

2001 年，GB/T 8566—1995 又被国家质量监督检验检疫总局新发布的《GB/T 8566—2001 信息技术 软件生存周期过程》所取代。GB/T 8566—2001 全面、系统地阐述了软件生命周期的 5 个主要过程（获取过程、供应过程、开发过程、运行过程、维护过程）、8 个支持过程（文档编制过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审核过程、问题解决过程）和 4 个组织过程（管理过程、基础设施过程、改进过程、培训过程）。

GB/T 8566—2001、GB/T 12504—1990、GB/T 12505—1990 是我国现阶段最重要的三个软件开发规范标准。

2．软件文档标准

前国家标准局 1988 年 1 月批准并发布的《GB 8567—1988 计算机软件产品开发文件编制指南》规定在一项软件开发过程中应该产生 14 种文件。

- A：可行性研究报告。
- B：项目开发计划。
- C：软件需求说明书。
- D：数据要求说明书。
- E：概要设计说明书。
- F：详细设计说明书。
- G：数据库设计说明书。
- H：用户手册。
- I：操作手册。
- J：模块开发卷宗。
- K：测试计划。
- L：测试分析报告。

- M：开发进度月报。
- N：项目开发总结报告。

其中管理人员主要使用的有项目开发计划、可行性研究报告、模块开发卷宗、开发进度月报和项目开发总结报告；开发人员主要使用的有项目开发计划、可行性研究报告、软件需求说明书、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划和测试分析报告；维护人员主要使用的有设计说明书、测试分析报告和模块开发卷宗。

第 16 章

软件设计概述

从功能上的划分来看，软件设计应该是软件设计师的工作。作为一名软件设计师，必须懂得软件设计的基本原则和理论，掌握基本的软件设计方法，具有丰富的软件设计经验。

16.1 软件设计基本原则

在软件设计过程中，必须遵循一些原则，例如信息隐蔽和模块独立性是两个最基本的原则。

16.1.1 信息隐蔽

传统的信息隐蔽起源于古老的隐写术。例如，在古希腊战争中，为了安全地传送军事情报，奴隶主剃光奴隶的头发，将情报纹在奴隶的头皮上，待头发长起后再派出去传送消息。我国古代也早有以藏头诗、藏尾诗、漏格诗以及绘画等形式，将要表达的意思和“密语”隐藏在诗文或画卷中的特定位置，一般人只注意诗或画的表面意境，而不会去注意或破解隐藏其中的密语。

在软件设计中，也有信息隐蔽原则，指的是：在概要设计时列出将来可能发生变化的因素，并在模块划分时将这些因素放到个别模块的内部。也就是说，每个模块的实现细节对于其他模块来说是隐蔽的，模块中所包含的信息（包括数据和过程）不允许其他不需要这些信息的模块使用。这样，在将来由于这些因素变化而需修改软件时，只需修改这些个别的模块，其他模块不受影响。信息隐蔽技术不仅提高了软件的可维护性，而且也避免了错误的蔓延，改善了软件的可靠性。现在信息隐蔽原则已成为软件工程中的一条重要原则。

16.1.2 模块独立性

软件设计中的模块独立性是指软件系统中每个模块只涉及软件要求的具体子功能，而模块间的接口简单。模块独立的概念是模块化、抽象、信息隐蔽和局部化概念的直接结果。

如何定义模块大小，Meyer 定义了如下 5 条标准。

- 模块的可分解性：如果一种设计方法提供了将问题分解成子问题的系统化机制，它就能降低整个系统的复杂性，从而实现一种有效的模块化解决方案。
- 模块的可组装性：如果一种设计方法使现存的（可复用的）设计构件能被组装成新系统，它就能提供一种不需要一切从头开始的模块化解决方案。

- 模块的可理解性：如果一个模块可以作为一个独立的单位（不用参考其他模块）被理解，那么它就易于构造和修改。
- 模块的连续性：如果对系统需求的微小修改只导致对单个模块，而不是整个系统的修改，则修改引起的副作用就会被最小化。
- 模块的保护性：如果模块内部出现异常情况，并且它的影响限制在模块内部，则错误引起的副作用就会被最小化。

一般采用两个准则度量模块的独立性，即模块间耦合和模块内聚。

耦合是模块之间的相对独立性（互相联系的紧密程度）的度量。模块之间的联系越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。

内聚是模块功能强度（一个模块内部各个元素彼此结合的紧密程度）的度量。一个模块内部各个元素之间的联系越紧密，则它的内聚性就越高；相对的，它与其他模块之间的耦合性就会减低，而模块独立性就越强。因此，模块独立性比较强的模块应是高内聚、低耦合的模块。

1．内聚

内聚是信息隐蔽功能的自然扩展。内聚的模块在软件过程中完成单一的任务，同程序其他部分执行的过程交互很少，简而言之，内聚模块（理想情况下）应该只完成一件事。在设计模块时应尽量争取高内聚。

一般模块的内聚性分为 7 种，如图 16-1 所示。

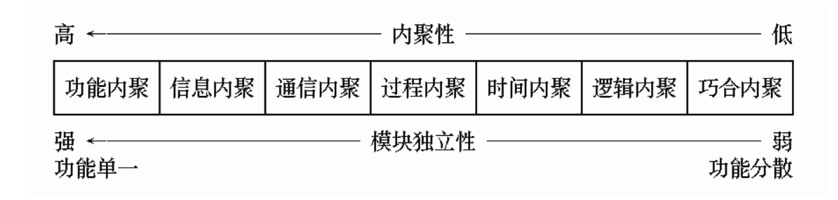


图 16-1　模块的内聚性

一般认为，巧合（偶然）、逻辑和时间上的聚合是低聚合性的表现；信息的聚合则属于中等聚合性；顺序的和功能的聚合是高聚合性的表现。表 16-1 所示为各类聚合性与模块各种属性的关系。

表 16-1　各类聚合性与模块各种属性的关系

	内 部 联 系	清　晰　性	可 重 用 性	可 修 改 性	可 理 解 性
巧合内聚	很差	差	很差	很差	很差
逻辑内聚	很差	很差	很差	很差	差
时间内聚	差	中	很差	中	中
过程内聚	中	好	差	中	中
通信内聚	好	中	中	中	中
信息内聚	好	好	中	好	好
功能内聚	好	好	好	好	好

1）功能内聚（Functional Cohesion）

一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作、紧密联系、不可分割的，则称该模块为功能内聚模块。它是内聚程度最高的，也是模块独立性最强的模块。

2）信息内聚（Informational Cohesion）

这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表）的，因此，它是一个信息内聚的模块。

信息内聚模块可以看成多个功能内聚模块的组合，并且达到信息的隐蔽。即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。

3）通信内聚（Communication Cohesion）

如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。

4）过程内聚（Procedural Cohesion）

使用流程图作为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成 3 个模块，这 3 个模块都是过程内聚模块。

5）时间内聚（Classical Cohesion）

时间内聚又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行，如初始化模块和终止模块。

6）逻辑内聚（Logical Cohesion）

这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。

7）巧合内聚（Coincidental Cohesion）

巧合内聚又称为偶然内聚。模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。

2．耦合

耦合是程序结构中模块相互关联的度量。耦合取决于各个模块间接口的复杂程度、调用模块的方式，以及哪些信息通过接口。

耦合的强度依赖于如下几个因素：

- 一个模块对另一个模块的调用。
- 一个模块向另一个模块传递的数据量。
- 一个模块施加到另一个模块的控制的多少。
- 模块之间接口的复杂程度。

一般模块之间可能的连接方式有 7 种，它们构成耦合性的 7 种类型，如图 16-2 所示。

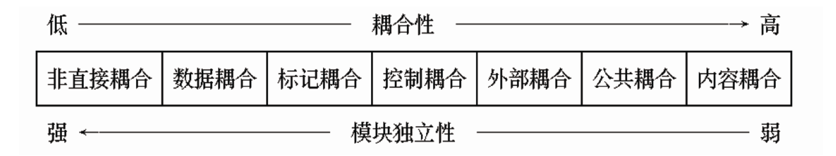


图 16-2　模块之间的耦合性

耦合是影响软件复杂程度的一个重要因素。在软件设计过程中，应尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。表 16-2 所示为各类耦合性与模块各种属性的关系。

表 16-2　各类耦合性与模块各种属性的关系

	对修改的敏感性	可 重 用 性	可 修 改 性	可 理 解 性
内容耦合	很强	很差	很差	很差
公共耦合	强	很差	中	很差
外部耦合	一般	很差	很差	中
控制耦合	一般	差	差	差
标记耦合	不一定	中	中	中
数据耦合	不一定	好	好	好
非直接耦合	好	好	好	好

1）非直接耦合（Nondirective Coupling）

如果两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的，这就是非直接耦合。这种耦合的模块独立性最强。

2）数据耦合（Data Coupling）

如果一个模块访问另一个模块时，彼此之间是通过简单数据参数（不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的，则称这种耦合为数据耦合。

3）标记耦合（Stamp Coupling）

如果一组模块通过参数表传递记录信息，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量。

4）控制耦合（Control Coupling）

如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。

5）外部耦合（External Coupling）

一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称为外部耦合。

6）公共耦合（Common Coupling）

若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。

公共耦合的复杂程度随耦合模块的个数增加而显著增加。若只是两模块间有公共数据环境，则公共耦合有两种情况：松散公共耦合和紧密公共耦合。

7）内容耦合（Content Coupling）

如果发生下列情形，两个模块之间就发生了内容耦合：

- 一个模块直接访问另一个模块的内部数据。
- 一个模块不通过正常入口转到另一模块内部。
- 两个模块有一部分程序代码重叠（只可能出现在汇编语言中）。
- 一个模块有多个入口。

3．深度、宽度、扇出与扇入

深度表示软件结构中控制的层数。如果层数过多则应考虑是否有许多管理模块过于简单，能否适当合并。

宽度是软件结构中同一个层次上的模块总数的最大值。一般说来，宽度越大系统越复杂。对宽度影响最大的因素是模块的扇出。

一个模块的扇出是指该模块直接调用的下级模块的个数。扇出大表示模块的复杂度高，需要控制和协调过多的下级模块；但扇出过小（例如总是 1）也不好。扇出过大一般是因为缺乏中间层次，应该适当增加中间层次的控制模块。扇出太小时可以把下级模块进一步分解成若干个子功能模块，或者合并到它的上级模块中去。

一个模块的扇入是指直接调用该模块的上级模块的个数。扇入大表示模块的复用程度高。

设计良好的软件结构通常顶层扇出比较大，中间扇出较小，底层模块则有大扇入。

应当注意，不应为了单纯追求深度、宽度、扇出与扇入的理想化而违背模块独立原则，分解或合并模块必须符合问题结构。

4．作用域和控制域

模块的作用域是指受该模块内一个判定影响的所有模块的集合。模块的控制域是指该模块本身及被该模块直接或间接调用的所有模块的集合。软件设计时，模块的作用域应在控制域之内，作用域最好是做出判定的模块本身及它的直属下级模块。

5．功能的可预测性

功能可预测是指对相同的输入数据能产生相同的输出。软件设计时应保证模块的功能是可以预测的。

16.2 结构化设计方法

结构化设计方法是在模块化、自顶向下逐层细化、结构化程序设计等程序设计技术的基础上发展起来的。该方法实施的过程如下：

- 总结出系统应有的功能，对一个功能，从功能完成的过程考虑，将各个过程列出，标识出过程转向和传递的数据。这样，可以将所有的过程都画出来。
- 细化数据流。确定应该记录的数据。
- 分析各过程之间的耦合关系，合理地进行模块划分以提高它们之间的内聚性。

16.2.1　系统结构图中的模块

模块就如同人的器官，具有特定的功能。人体中最出色的模块设计之一是手，手只有几种动作，却能做无限多的事情。人体中最糟糕的模块设计之一是嘴巴，嘴巴将最有价值但毫不相干的几种功能，如吃饭、说话、亲吻，混为一体，使之无法并行处理。

在系统结构图中，不能再分解的底层模块称为原子模块。如果一个软件系统的全部实际加工都由原子模块来完成，而其他所有非原子模块仅仅执行控制或协调功能，这样的系统就是完全因子分解的系统。如果系统结构图是完全因子分解的，就是最好的系统。但实际上，这只是力图达到的目标，大多数系统做不到完全因子分解。

一般来说，结构图中可能出现如图 16-3 所示的 4 种类型的模块。

- 传入模块：如图16-3（a）所示，从下属模块取得数据，经过某些处理，再将其传送给上级模块。它传送的数据流叫作逻辑输入数据流。
- 传出模块：如图16-3（b）所示，从上级模块取得数据，进行某些处理，传送给下属模块。它传送的数据流叫作逻辑输出数据流。
- 变换模块：如图16-3（c）所示，从上级模块取得数据，进行特定处理后，送回原上级模块。它加工的数据流叫作变换数据流。
- 协调模块：如图16-3（d）所示，对其下属模块进行控制和管理的模块。在一个好的系统结构图中，协调模块应在较高层出现。

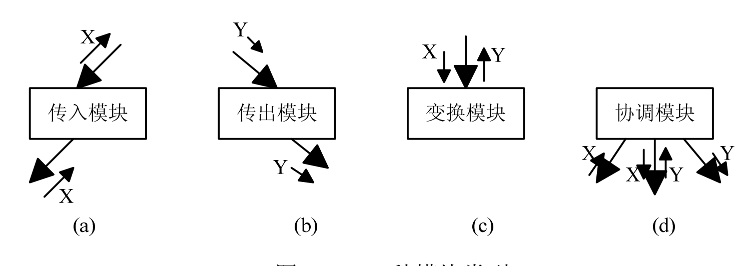


图 16-3　4 种模块类型

值得注意的是，结构图着重反映的是模块间的隶属关系，即模块间的调用关系和层次关系。它和程序流程图（常称为程序框图）有着本质的差别。程序流程图着重表达的是程序执行的顺序及执行顺序所依赖的条件。结构图则着眼于软件系统的总体结构，它并不涉及模块内部的细节，只考虑模块的作用，以及它和上、下级模块的关系。而程序流程图则用来表达执行程序的具体算法。

没有学过软件开发技术的人，一般习惯于使用流程图编写程序，往往在模块还未做划分、程序结构的层次尚未确定以前，便急于用流程图表达他们对程序的构想。这就像造一栋大楼，在尚未决定建筑面积和楼层有多少时，就已经开始砌砖了。这显然是不合适的。

Adele Goldberg 在 *Succeeding with Objects* 中叙述了一位犹太教教士在新年伊始的宗教集会上讲述的故事：

一位教士登上一列火车，由于他经常乘坐这辆车，因此列车长认识他。教士伸手到口袋中掏车票，但没有找到，他开始翻他的行李。列车长阻止了他："教士，我知道您肯定有车票。现在别急着找。等找到后再向我出示。"但教士仍在找那张车票。当列车长再次见到他时，教士说："你不明白。我知道你相信我有车票，但……我要去哪里呢？"

有太多项目失败就是因为它们没有明确的目标就开始了。

在结构化分析和设计技术中，通常存在着两种典型的问题类型：变换型问题和事务型问题。它们的数据流图和结构图都有明显的特征。下面分别讨论它们的数据流图形态及其映射成结构图的过程。

结构图（Structured Charts，简称 SC）是准确表达程序结构的图形表示方法，它能清楚地反映出程序中各模块间的层次关系和联系。与数据流图反映数据流的情况不同，结构图反映的是程序中控制流的情况，如图 16-4 所示为某大学教务管理系统的结构图。

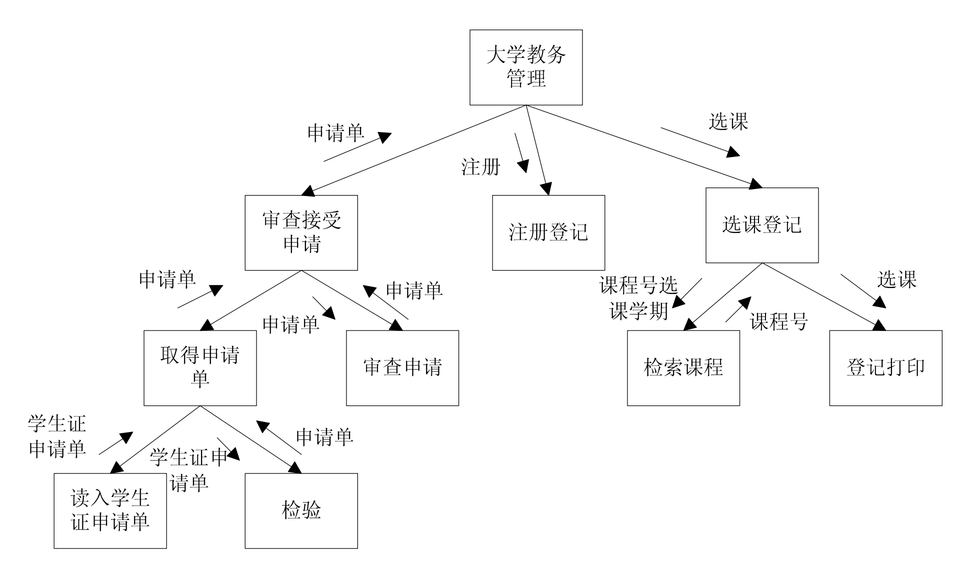


图 16-4　大学教务管理系统结构图

16.2.2　系统结构图中的主要成分

系统结构图中有如下主要成分。

1．模块

以矩形框表示，框中标有模块的名字。对于已定义（或者已开发）的模块，则可以用双纵边矩形框表示，如图 16-5 所示。

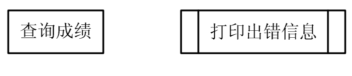


图 16-5　模块的表示

2．模块间的调用关系

两个模块，一上一下，以箭头相连，上面的模块是调用模块，箭头指向的模块是被调用模块，如图 16-6 所示，模块 A 调用模块 B。在一般情况下，箭头表示的连线可以用直线代替。

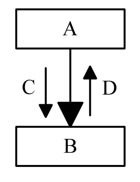


图 16-6　模块的调用关系及信息传递关系的表示

3．模块间的通信

模块间的通信以表示调用关系的长箭头旁边的短箭头表示，短箭头的方向和名字分别表示调用模块和被调用模块之间信息的传递方向和内容，如图 16-6 所示，首先模块 A 将信息 C 传给模块 B，经模块 B 加工处理后的信息 D 再传回给 A。

4．辅助控制符号

当模块 A 有条件地调用模块 B 时，在箭头的起点标以菱形。模块 A 反复地调用模块 D 时，另加一环状箭头，如图 16-7 所示。

在结构图中条件调用所依赖的条件和循环调用的循环控制条件通常都无须注明。

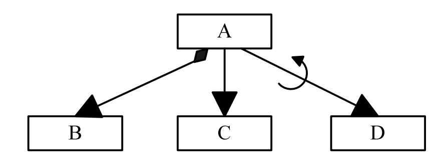


图 16-7　条件调用和循环调用的表示

16.2.3　常用的系统结构图

常用的系统结构图有如下几种。

1．变换型系统结构图

在数据处理问题中，我们通常会遇到这样一类问题，即从（程序）“外部”（如从键盘、磁盘文件等）取得数据，对取得的数据进行某种变换，然后再将变换得到的数据传回给“外部”。其中，取得数据这一过程称为传入信息（数据）流程，变换数据的过程称为变换信息（数据）流程，传回数据的过程称为传出信息（数据）流程，如图 16-8 所示。

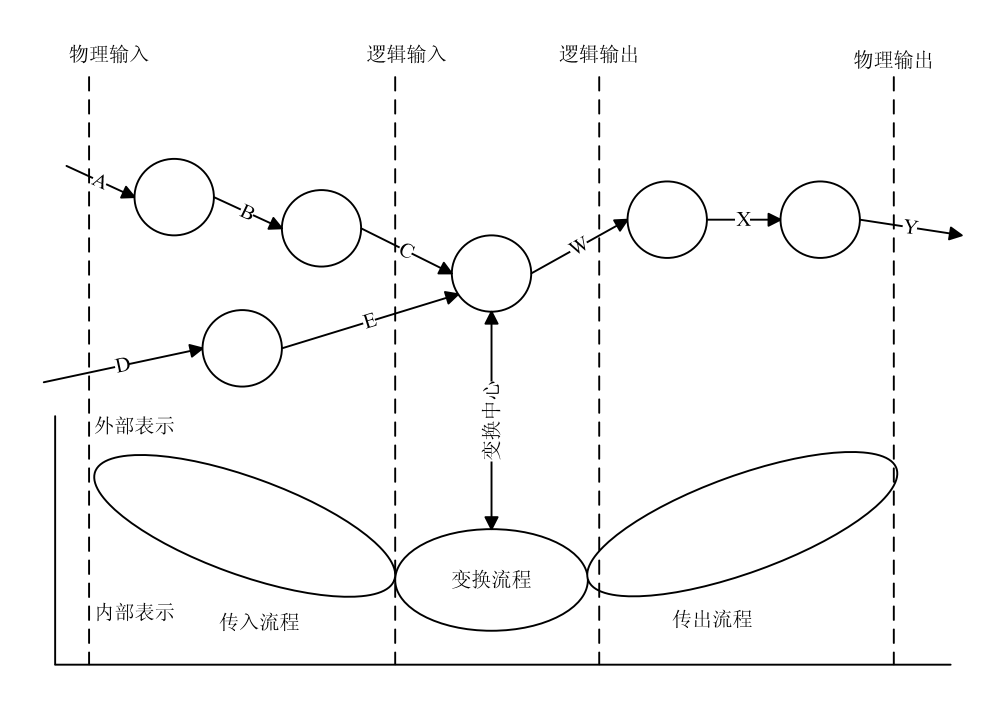


图 16-8　变换型问题

当数据流图或其中某一段数据流表现出上述特征时，该数据流图或该段数据流图表示的就是一个变换型问题。完成数据变换的处理单元称为变换中心。变换型问题数据流图基本形态及其对应的基本结构图分别如图 16-9（a）和图 16-9（b）所示。

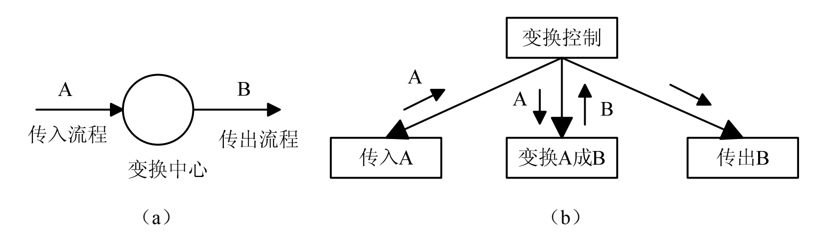


图 16-9　基本变换型问题数据流图及其结构图

根据图 16-9 表示的基本映射关系所得到的如图 16-8 所示变换型问题的结构图如图 16-10 所示。“变换控制”模块首先获得控制，然后控制沿着结构到达底层的“传入 A”模块，物理输入数据 A 由“传入 A”模块读入后，从底层逐步向上传送。在传送过程中，数据经“变换 A 成 B”、“变换 B 成 C”等模块的预处理，逐渐变换成纯粹的逻辑输入 C、E。接着在“变换控制”模块的控制下，将逻辑输入经变换中心模块“变换 C 和 E 成 W”处理后，变换成逻辑输出 W，再从顶层逐步向下传送。在这一传送过程中，数据经“变换 W 成 X”、“变换 X 成 Y”等模块的后处理，逐渐变换成适当的输出形式，最后由“传出 Y”模块完成物理输出。

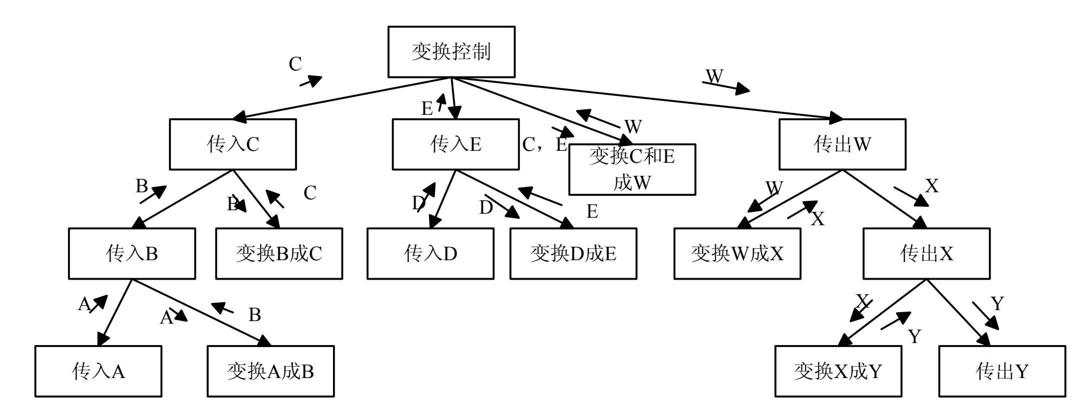


图 16-10　变换型问题结构图

2．事务型系统结构图

在实际中，我们还常常会遇到另一类问题，即通常在接受某一项事务后，根据事务的特点和性质，选择分派给它一个适当的处理单元，然后给出结果，如图 16-11 所示。

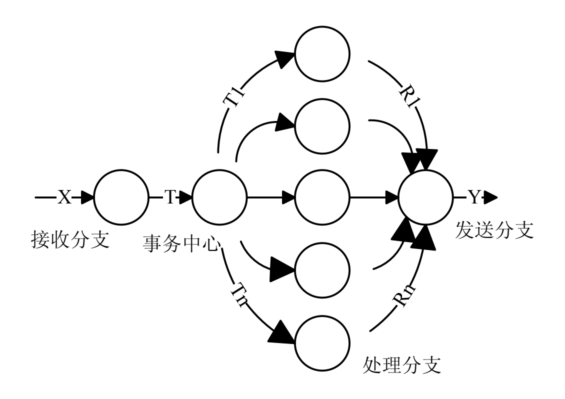


图 16-11　事务型问题

这类问题就是事务型问题。它的特点是，数据沿着接收分支把外部信息（数据）转换成一个事务项，然后计算该事务项的值，并根据它的值从多条数据流中选择其中的某一条数据流。发出多条数据流的处理单元称为事务中心。这类问题的典型结构图如图 16-12 所示。事务控制模块按所取得事务的类型，选择调用某一个处理事务模块。

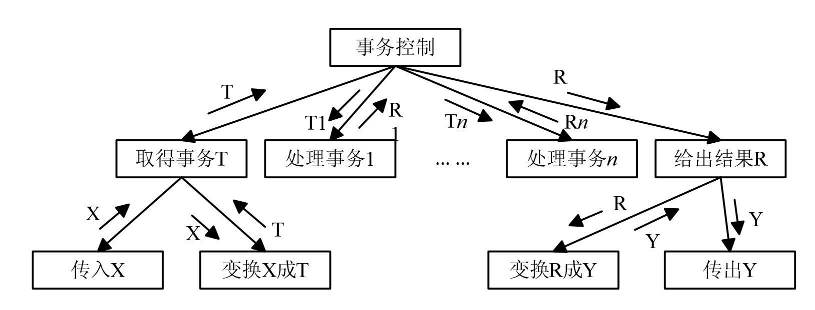


图 16-12　事务型问题结构图

16.3　面向对象设计

为了讨论面向对象的技术和方法，必须首先明确什么是“面向对象”。为什么要讨论面向对象的方法？什么是对象？

16.3.1　面向对象的概念

对于上述这些问题，有许多不同的看法。其中 Booch、Coad/Yourdon 和 Jacobson 的方法在面向对象软件开发界得到了广泛的认可。特别值得一提的是统一建模语言（Unified Modeling Language，UML），该方法结合了 Booch、OMT 和 Jacobson 方法的优点，统一了符号体系，并从其他的方法和工程实践中吸收了许多经过实际检验的概念和技术。

1．对象

对象是建立面向对象程序所依赖的基本单元。用更专业的语言来说，所谓对象就是一种代码的实例，这种代码执行特定的功能，具有自包含或者封装的性质。这种封装代码通常叫作类、对象类、模块或者在不同编程语言中所应用的其他名称。以上这些术语在含义上稍微有些不同，但它们都是代码的集合。

正如上面提到的那样，对象本身是类或者其他数据结构的实例。这就是说，现有的代码起到了创建对象的模板作用。执行特定功能的代码只需要编写一次却可以被引用多次。每一种对象具有自己的标识，也就是令对象相互区别的对象名称。

对象并不是类的实际拷贝。每一对象都有自己的名称空间，在这种名称空间中保存自己的标识符和变量，但是对象要引用执行函数的原有代码。

“封装”的对象具有自己的函数，这种函数被称为“方法”，而对象的变量则被称为属性。当对象内部定义了属性时，它们通常不能扩展到实例以外。假设现有一个类叫 autocar（汽车），同时又创建了两个对象实例 sedan（小轿车）和 bus（客车），那么给 sedan 设置的值就不会影响到 bus 内部的值。autocar 自身内部的变量却永远不会得到定义，因为 autocar 类只是一种模板。

在特定的场合下，有些函数确实会影响类而不是由类所创建的对象。类属性指的是专门设计来保留对象之间所用的值。类方法则用来定义和跟踪类属性。

某些编程语言可以让用户调用类的函数而不是创建整个实例。如果函数被分配以标识符（或者句柄），在某些情况下它们可以被视作具有自身权限的对象。不过，在大多数的情况下，函数只是用来实现某种结果的方法。

2．类

类是对象的抽象定义，是一组具有相同数据结构和相同操作的对象的集合。类的定义包括一组数据属性和在数据上的一组合法操作。类定义可以视为一个具有类似特性与共同行为的对象的模板，可用来产生对象。在一个类中，每个对象都是类的实例（Instance），它们都可使用类中提供的函数。一个对象的状态则包含在它的实例变量中。

3．继承

继承是使用已存在的定义作为基础建立新定义的技术。新类的定义可以是现存类所声明的数据、定义与新类所增加的声明的组合。新类复用现存类的定义，而不要求修改现存类。因为这种类的一部分已经实现和测试，故开发费用较少。现存类可当作父类（泛化类、基类或超类）来引用，则新类相应地可当作子类（特化类、子女类或派生类）来引用。

16.3.2　面向对象分析方法

面向对象分析（Object-Oriented Analyst，OOA）方法是对问题空间的理解和分析，分析阶段通过类或对象的认定，确定类之间（或对象间）的关系，然后对它们的属性、所提供的方法和所需要的方法进行描述，并按照它们之间的关系进行组织，得到类（或对象）结构。OOA 建立在信息模拟（ER 图和语义数据模型）和面向对象程序设计语言的概念基础上，如图 16-13 所示，从信息模拟中吸取了属性、关系、结构及对象作为问题域中某些事物的实例的表示方法等概念，从面向对象程序设计语言中吸取了属性和方法的封装，属性和方法作为一个不可分割的整体，以及各类结构和继承等概念。

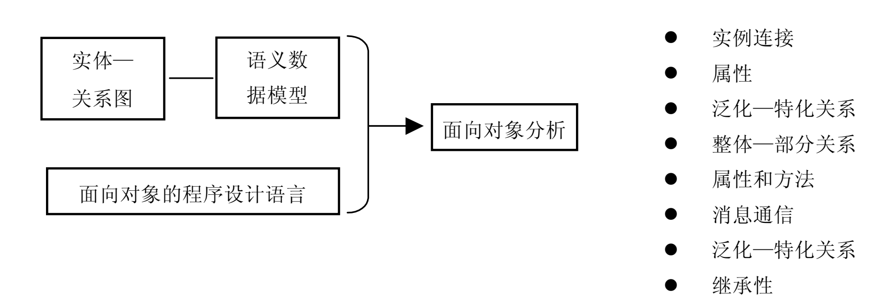


图 16-13　OOA 的形成

OOA 基于如下几点：

- 在分析和规格说明的总体框架中贯穿结构化方法，如对象和属性，整体和部分，类和成员等。
- 用消息进行用户和系统之间，以及系统中实例间的相互通信。
- 在识别每个部分所提供方法的总体框架中对性能进行分类。

16.3.3　面向对象设计

分析阶段主要是明确用户的功能需求，即满足用户所需的系统部件及其结构。设计阶段则主要是确定实现用户需求的方法，即怎样做才能满足用户需求，并构造出系统的实现

蓝图。面向对象设计（Object-Oriented Design，OOD）也是如此，只不过是引入了一些面向对象的概念和原则，用以指导设计工作。OOD 首先从 OOA 的结果开始，并将其从问题域映射到实现域；为满足实现的需要，还要增加一些类、结构及属性和服务，并对原有类及属性进行调整。此外，还要完成应用控制、人机交互界面的设计等。

面向对象的设计方法主要有：Booch 方法、OMT 方法和 OOSE 方法，而对这些方法进行整合又形成了 UML，关于 UML 的详细情况可参看第 18 章。在此主要介绍面向对象的设计原则。面向对象的设计原则包括如下 7 条。

- 单一职责原则：设计目的单一的类。
- 开放-封闭原则：对扩展开放，对修改封闭。
- 李氏（Liskov）替换原则：子类可以替换父类。
- 依赖倒置原则：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程。
- 接口隔离原则：使用多个专门的接口比使用单一的总接口要好。
- 组合重用原则：要尽量使用组合，而不是继承关系达到重用目的。
- 迪米特（Demeter）原则（最少知识法则）：一个对象应当对其他对象有尽可能少的了解。

16.4 用户界面设计

在人和机器的互动过程中，有一个层面，即我们所说的界面（interface）。从心理学意义来分，界面可分为感觉（视觉、触觉、听觉等）和情感两个层次。用户界面设计是屏幕产品的重要组成部分。界面设计是一个复杂的有不同学科参与的工程，认知心理学、设计学、语言学等在此都扮演着重要的角色。用户界面设计的三大原则是：置界面于用户的控制之下；减少用户的记忆负担；保持界面的一致性。

- 置用户于控制之下：具体来说就是以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式、提供灵活的交互、允许用户交互可以被中断和撤销、当技能级别增长时可以使交互流水化并允许定制交互、使用户隔离内部技术细节、设计应允许用户和出现在屏幕上的对象直接交互。
- 减少用户的记忆负担：具体来说就是减少对短期记忆的要求、建立有意义的默认、定义直觉性的捷径、界面的视觉布局应该基于真实世界的隐喻、以不断进展的方式提示信息。
- 保持界面的一致性：具体来说就是允许用户将当前任务放入有意义的语境、在应用系列内保持一致性、如果过去的交互模型已经建立了用户期望，除非有不得已的理由，否则不要改变它。

16.5 设计评审

在开发时期的每个阶段，特别是设计阶段结束时都要进行严格的技术评审，尽量不让错误传播到下一个阶段。设计评审一般采用评审会议的形式来进行。

软件设计评审流程如图 16-14 所示。

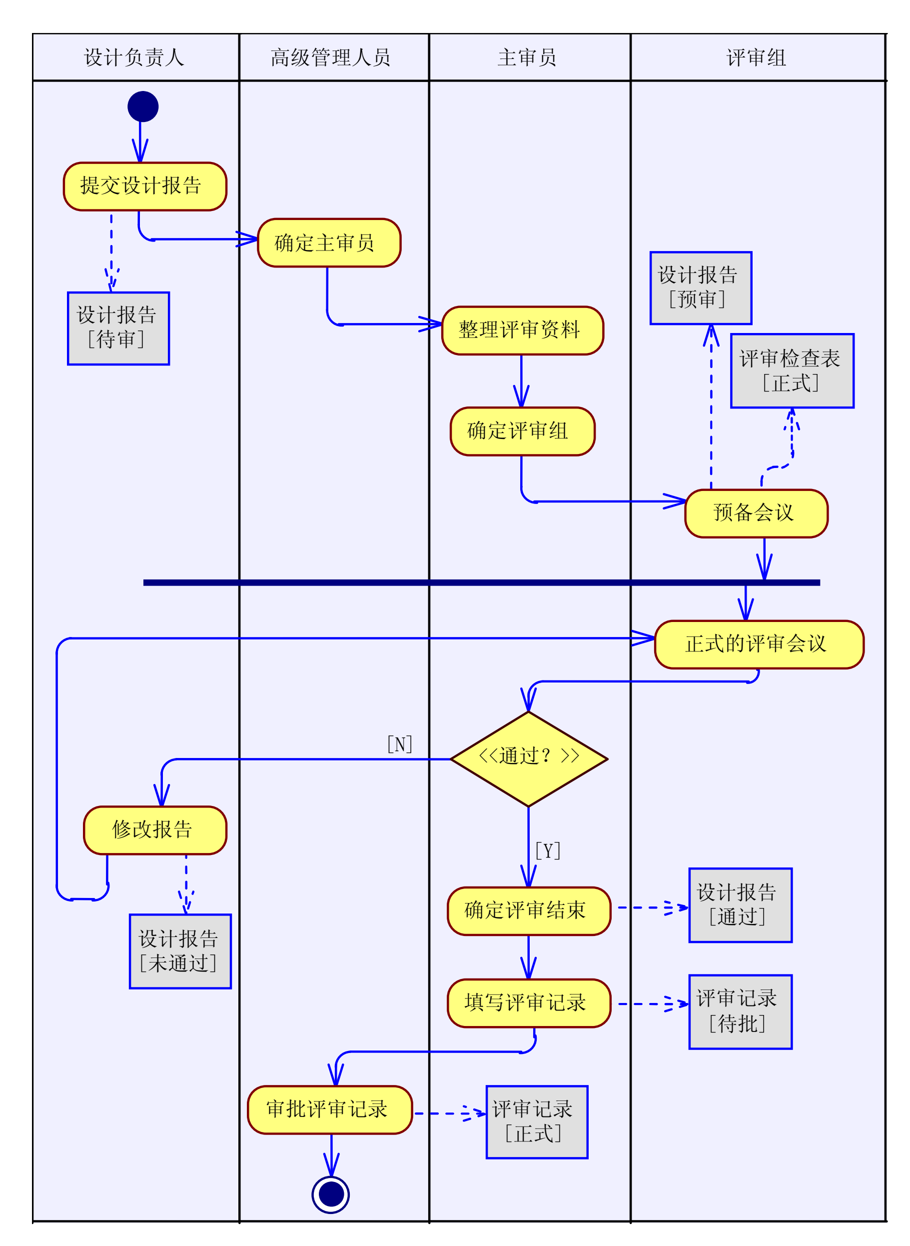


图 16-14　软件设计评审流程图

设计负责人的职责：一般工程设计均由软件公司选派设计负责人，设计负责人承担该项工程的全部设计管理任务，对设计质量、进度及各单项设计间的组织协调等全面负责，对各单项工程之间的衔接、协调和总体方案质量负主要责任，并负责编写总说明，汇编总概算。

- 高级管理人员的职责：确定主审员、审批评审记录。
- 主审员的职责：在评审会前提出项目的书面评审意见、确定评审组、确定评审结果并填写评审记录。
- 评审组的职责：专业评审组评委表决通过项目初评结论并报综合评审会议；通过设计报告。

第 17 章

数据流图设计

软件设计师不仅要具备高水平的程序编制能力，而且要熟练掌握软件设计的方法和技术，具备一定的软件设计能力。在软件设计师考试中几乎每次都会考数据流图设计题，所以要求考生熟练地掌握数据流图的设计。

17.1 数据流图

数据流图简称 DFD，是描述数据处理过程的一种图形工具。数据流图从数据传递和加工的角度，以图形的方式描述数据在系统流程中流动和处理的移动变换过程，反映数据的流向、自然的逻辑过程和必要的逻辑数据存储。

17.1.1 数据流图基本图形符号

数据流图采用 4 种基本的图形符号，如表 17-1 所示。

表 17-1 数据流图基本符号

符　　号	名　　称	说　　明
○	加工	在圆中注明加工的名字与编号
→	数据流	在箭头边给出数据流的名称与编号，注意不是控制流
═	数据存储文件	文件名称为名词或名词性短语
□	数据源点或汇点	在方框中注明数据源或汇点的名称

1．加工

加工用圆或椭圆描述，又称为数据处理，表示输入数据在此进行变换产生输出数据，以数据结构或数据内容作为加工对象。加工的名字通常是一个动词短语，简明扼要地表明要完成的加工。

2．数据流

用箭头描述，由一组固定的数据项组成，箭头方向表示数据的流向，作为数据在系统内的传输通道。它们大多是在加工之间传输加工数据的命名通道，也有在数据存储文件和加工之间的非命名数据通道。虽然这些数据流没有命名，但其连接的加工和文件的名称，以及流向可以确定其含义。

同一数据流图上不能有同名的数据流。如果有两个以上的数据流指向一个加工，或从一个加工中输出两个以上的数据流，则这些数据流之间往往存在一定的关系。其具体的描述如图 17-1 所示，其中“*”表示相邻之间的数据流同时出现，“ ⊕ ”表示相邻之间的数据流只取其一。

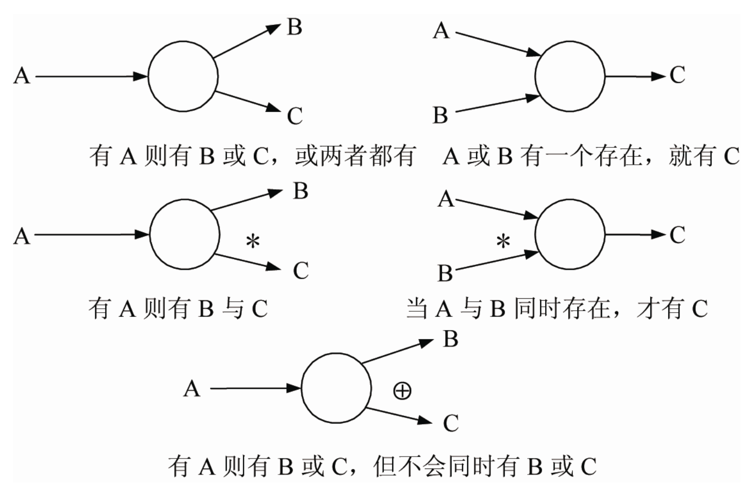


图 17-1　数据流

3．数据存储文件

用双杆描述，在数据流图中起保存数据的作用，又称为数据存储或文件，可以是数据库文件或任何形式的数据组织。流向数据存储的数据流可以理解为写入文件或查询文件，从数据存储流出的数据流可以理解为从文件读数据或得到查询结果。

4．数据源点或终点

用方框描述，表示数据流图中要处理数据的输入来源或处理结果要送往的地方，在图中仅作为一个符号，并不需要以任何软件的形式进行设计和实现，是系统外部环境中的实体，故称为外部实体。它们作为系统与系统外部环境的接口界面，在实际的问题中可能是人员、组织、其他软/硬件系统等。一般只出现在分层数据流的顶层图中。

17.1.2　数据流图设计要略

有时为了增加数据流图的清晰性，防止数据流的箭头线太长，减少交叉绘制数据流条数，一般在一张图上可以重复同名的数据源点、终点与数据存储文件。如某个外部实体既是数据源点又是数据汇点，可以在数据流图的不同的地方重复绘制。在绘制时应该注意如下要点。

- 自外向内，自顶向下，逐层细化，完善求精。

- 保持父图与子图的平衡。也就是说，父图中某加工的输入/输出数据流必须与它的子图的输入/输出数据流在数量和名字上相同。
- 保持数据守恒。也就是说，一个加工所有输出数据流中的数据必须能从该加工的输入数据流中直接获得，或者是通过该加工能产生的数据。
- 加工细节隐蔽。根据抽象原则，在画父图时，只需画出加工和加工之间的关系，而不必画出各个加工内部的细节。
- 简化加工间的关系。在数据流图中，加工间的数据流越少，各个加工就越相对独立，所以应尽量减少加工间输入/输出数据流的数目。
- 均匀分解。应该使一个数据流中的各个加工分解层次大致相同。
- 适当地为数据流、加工、文件、源/宿命名，名字应反映该成分的实际意义，避免空洞的名字。
- 忽略枝节。应集中精力于主要的数据流，而暂不考虑一些例外情况、出错处理等枝节性的问题。
- 表现的是数据流而不是控制流。数据流图与传统的程序流程图不同，数据流图是从数据的角度来描述一个系统的，而流程图则是从对数据加工的角度来描述系统的。数据流图中的箭头是数据流，而流程图中的箭头则是控制流，它表达的是程序执行的次序。数据流图适合于宏观地分析一个组织的业务概况，而程序流程图只适合于描述系统中某个加工的执行细节。
- 每个加工必须既有输入数据流，又有输出数据流；在整套数据流图中，每个文件必须既有读文件的数据流又有写文件的数据流，但在某一张子图中可能只有读、没有写，或者只有写、没有读。

17.1.3　数据字典

数据字典的任务就是对数据流图中出现的所有被命名的图形元素在数据字典中作为一个词条加以定义，使得每个图形元素的名称都有一个确切的解释。

数据字典描述的内容包括数据流、数据文件、加工逻辑、源（汇）点及数据元素等词条的描述。在数据流和数据文件词条的数据字典描述中包含一定的数据结构，对于数据结构常用的描述是定义式。表 17-2 所示为数据结构定义式可能出现的符号。

表 17-2　数据结构定义式可能出现的符号

符　　号	含　　义	举 例 说 明
=	被定义为	
+	与	x=a+b，表示 x 由 a 和 b 组成
[…，…]或[…|…]	或	x=[a，b]，x=[a|b]，表示 x 由 a 或由 b 组成
{…}	重复	x={a}，表示 x 由 0 个或多个 a 组成
(…)	可选	x=(a)，表示 a 可在 x 中出现，也可以不出现

在数据字典中有 4 种类型的条目。

1．数据项条目

数据项条目给出了某个数据单项的定义，通常为数据项的值类型、允许的取值范围等。

2．数据流条目

数据流条目给出某个数据流的定义，它通常是列出该数据流的各组成数据项。有些数据流的组成比较复杂，可以采用自顶向下分解的方式将它表示成更低层次的组合，一直分解到每个与项目有关的人都清楚其准确含义时为止。

由低的数据元素（或称分量）组成更复杂的数据的方式有如下几种。

- 顺序：即以确定次序连接两个或多个分量。
- 选择：即从两个或多个可能的元素中选取一个。
- 重复：即把指定的分量重复零次或多次。
- 可选：即一个分量是可有可无的（重复零次或多次）。

3．文件条目

文件条目给出某个文件的定义，通常也是列出其记录的组成数据项。此外，还可以指出文件的组织方式，如按单号递增次序排列等。

4．加工条目

加工条目是对数据流图中每一个不能再分解的基本加工的精确说明。

说明中应精确描述用户要求某个加工做什么，包括加工的激发条件、加工逻辑、优先级、执行频率和出错处理等。其中加工逻辑是最基本的部分，它描述了输入数据流、输入文件与输出数据流、输出文件之间的逻辑关系。常用的加工逻辑描述方法有 3 种：结构化语言、判定表和判定树。

17.1.4　分层数据流图

为了表达较为复杂问题的数据处理过程，用一个数据流图往往不够。一般按问题的层次结构进行逐步分解，并以分层的数据流图反映这种结构关系。

根据层次关系一般将数据流图分为顶层数据流图、中间数据流图和底层数据流图，除顶层图外，其余分层数据流图从 0 开始编号。对任何一层数据流图来说，称它的上层数据流图为父图，在它的下一层的数据流图为子图。

顶层数据流图只含有一个加工，表示整个系统；输入数据流和输出数据流为系统的输入数据和输出数据，表明了系统的范围，以及与外部环境的数据交换关系。

底层数据流图是指其加工不能再分解的数据流图，其加工称为“原子加工”。

中间数据流图是对父层数据流图中某个加工进行细化，而它的某个加工也可以再次细化，形成子图。中间层次的多少，一般视系统的复杂程度而定。

17.1.5　分层数据流图的解答要点

通常设计分层数据流图需要注意如下几点。

1．父图与子图的平衡

任何一个数据流子图必须与它上一层父图的某个加工对应，二者的输入数据流和输出数据流必须保持一致，此即父图与子图的平衡。父图与子图的平衡是数据流图中的重要性质，保证了数据流图的一致性，便于分析人员阅读和理解。

在父图与子图平衡中，数据流的数目和名称可以完全相同；也可以在数目上不相等，

但是可以借助数据字典中数据流描述，确定父图中的数据流是由子图中几个数据流合并而成的，也即子图对父图中加工和数据流同时进行分解，因此也属于父图与子图的平衡，如图 17-2 所示。

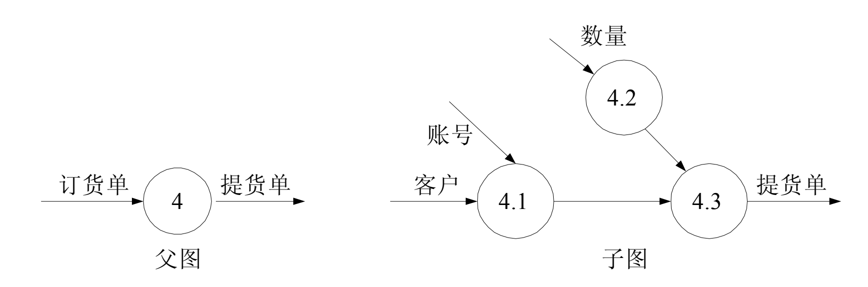


图 17-2　父图与子图的平衡

2．局部数据存储的隐蔽性

当某层数据流图中的数据存储不是父图中相应加工的外部接口，而只是本图中某些加工之间的数据接口时，那么这些数据存储为局部数据存储。

为了强调局部数据存储的隐蔽性，一般情况下，局部数据存储只有作为某些加工的数据接口或某个特定加工的输入和输出时，才画出来。即按照自顶向下的分析方法，某数据存储首次出现时只与一个加工有关，那么这个数据存储应该作为与之关联加工的局部数据存储，在该层数据流子图中不必画出，而在该加工的子图中画出，除非该加工为原子加工。

3．输入/输出的平衡性

每个加工必须有输入数据流和输出数据流，反映此加工的数据来源和加工变换结果。

17.2　系统流程图

系统流程图又称为事务流程图，是计算机事务处理应用进行系统分析时常用的一种描述方法，借助图形符号来表示系统中各元素。它描述计算机事务处理中从数据输入开始到获得输出为止，各个处理工序的逻辑过程。

17.2.1　系统流程图基本处理

系统流程图一般含有变换、合并、划分、分类、更新 5 种基本的处理。

1．变换

交换是指把输入单据变换成磁盘文件，或把磁盘文件变换成输出单据，或把某一磁盘文件的内容由一个介质文件传送到另一个介质文件。

一般在进行输入变换的同时，还可对输入的数据进行形式性的逻辑检查，如数据输入错误、含有非法字符、数据类型错误等。另外一个方面，是对输入的数据结合外部文件进行合法性检查，如数据值不存在、数据值的越界等。

2．合并

合并是指把多个文件合并为一个文件。

3．划分

划分是合并的逆操作，将合并工序的输入文件与输出文件对调即可。

4．分类（排序）

分类（排序）是按指定的键（关键字）以升序或降序改变原文件的记录排列顺序。分类也可和输入或输出操作一起进行。

5．更新

更新是将多个文件作为输入，根据关键项目进行对照，对文件内容进行修正、删除、增加等改写工作。一般更新的内容先要写入一个临时文件，在一定的工作时间后（一般在系统中都会进行说明，如一个月），为了提高系统的处理效率，一般要将该文件进行全部的清理或者部分清理。

17.2.2　系统流程图解题要点

系统处理流程是事务之间相互关系及处理的先后次序的表示，数据是事务的处理依据，也是事务的处理结果，因此可以从处理和数据两个角度出发，对系统流程图进行分析与问题的解答。

1．处理角度

根据处理在流程图中的作用及位置，一般将处理分成系统目标处理和基本处理两大类。

- 系统目标处理。在系统流程图中一般要对系统所需要完成的目标进行文字性的定义和描述，那么在流程图中应该有一个与之对应的处理，该处理能够覆盖系统所给定的目标。
- 基本处理。流程图中除覆盖系统目标的处理外，还有一些为系统目标处理服务的基本处理，主要包括两个方面：一是为了处理的正确性，设计一些处理，以检查输入数据的数据项及数据的值域，以及检查数据的正确性和一致性等；二是为了处理的效率，如提高处理速度、减少文件冗余度等引进了一些处理。

2．数据角度

使用数据是处理的依据，产生数据是处理的结果，所以数据的使用和产生应该与处理相互匹配。对于流程图中的数据，应该主要注意如下几个方面的描述。

- 最初的输入数据与最终的输出数据：它确定了系统的处理目标，以及从输入到输出之间数据的演变过程。根据数据的演变与流程，关于从输入到输出应有哪些数据就比较清楚了，其作用也可以从演变方面了解。
- 数据存储要求：在数据演变的过程中，一些数据经多个“处理”后得到最后结果，每加工处理一次就产生一个新数据，通过对这些数据作用的分析，确定哪些数据应作为文件形式出现，哪些是中间使用的临时数据，这样就能得出各数据的存储要求。
- 数据结构设计：输入/输出数据的结构与系统的问题有关，而中间数据的结构除与输入/输出数据有关外，还与处理有关。在设计的过程中，应该考虑各种数据之间的联系，保证数据的一致性。

第 18 章

UML 分析与设计

统一建模语言（Unified Modeling Language，UML）已经日益成为建模标准，应用越来越广泛，现在已经成为软件分析与设计建模的标准。所以 UML 分析与设计的试题也成为每次软件设计师考试中必考的题型。

18.1 UML 概述

20 世纪 80—90 年代，面向对象的分析与设计（OOA&D）方法获得了长足的发展，而且相关的研究也十分活跃，涌现了大量的方法学，据不完全统计，最多的时候高达 50 多种。其中最有代表性的当数 Booch（Grady Booch）、OMT（Jim Rumbaugh）、OOSE（Ivar Jacobson）3 种方法，而 UML 正是这 3 位“大师”联手，共同打造而成的，现已成为标准的建模语言。

18.1.1 UML 是什么

统一建模语言（Unified Modeling Language，UML）是用于系统的可视化建模语言，尽管它常与建模 OO 软件系统相关联，但由于其内建了大量扩展机制，还可以应用于更多的领域中，如工作流程、业务领域等。

- UML是一种语言：UML在软件领域中的地位与价值就像“1、2、3、+、−、…”等符号在数学领域中的地位一样。它为软件开发人员提供了一种用于交流的词汇表，是一种用于软件蓝图的标准语言。
- UML是一种可视化语言：UML只是一组图形符号，它的每个符号都有明确语义，是一种直观、可视化的语言。
- UML是一种可用于详细描述的语言：UML所建的模型是精确的、无歧义和完整的，因此适合于对所有重要的分析、设计和实现决策进行详细描述。
- UML是一种构造语言：UML虽然不是一种可视化的编程语言，但其与各种编程语言直接相连，而且有较好的映射关系，这种映射允许进行正向工程、逆向工程。
- UML是一种文档化语言：它适合于建立系统体系结构及其所有的细节文档。

18.1.2 UML 结构

UML 的结构如图 18-1 所示。

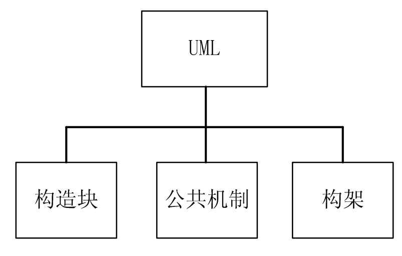


图 18-1　UML 结构示意图

1．构造块

构造块是基本的 UML 建模元素、关系和图。

- 建模元素：包括结构元素（类、接口、协作、用例、活动类、组件、结点等）、行业元素（交互、状态机）、分组元素（包）、注解元素。
- 关系：包括关联关系、依赖关系、泛化关系、实现关系。
- 图：在UML 2.0中包括14种不同的图。用例图、类图、对象图、构件图、部署图、状态图、顺序图、活动图、通信图（协作图）、组成结构图、交互概况图、定时图、制品图、包图，UML1.x中只包含前9种图。

另外，要注意的是，各种书籍对上述名称的翻译也不同，例如，有些书籍把“元素”直译为“事物”等。

2．公共机制

公共机制是指达到特定目标的公共 UML 方法，主要包括规格说明、修饰、公共分类和扩展机制 4 种。

- 规格说明：规格说明是元素语义的文本描述，它是模型真正的“肉”。
- 修饰：UML为每一个模型元素设置了一个简单的记号，还可以通过修饰来表达更多的信息。
- 公共分类：包括类元与实体（类元表示概念，而实体表示具体的实体）、接口和实现（接口用来定义契约，而是实现就是具体的内容）两组公共分类。
- 扩展机制：包括约束（添加新规则来扩展元素的语义）、构造型（用于定义新的UML建模元素）、标记值（添加新的特殊信息来扩展模型元素的规格说明）。

3．构架

UML 对系统构架的定义是：系统的组织结构，包括系统分解的组成部分、它们的关联性、交互、机制和指导原则。这些提供系统设计的信息，而具体来说，就是指 5 个系统视图。

- 逻辑视图：以问题域的语汇组成的类和对象集合。
- 进程视图：可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例。
- 实现视图：对组成基于系统的物理代码的文件和组件进行建模。
- 部署视图：把组件部署到一组物理的、可计算的节点上。
- 用例视图：最基本的需求分析模型。

18.1.3　UML 的主要特点

UML 的主要特点如下。

- UML统一了Booch、OMT、OOSE和其他面向对象方法的基本概念和符号，同时汇集了面向对象领域中很多人的思想，是优秀的面向对象方法，是在丰富的计算机科学实践中总结而成的。
- 目前UML是最先进、实用的标准建模语言，而且还在不断发展进化之中。
- UML是一种建模语言而不是一种方法，其中并不包括过程的概念，其本身是独立于过程的，用户可以在使用过程中使用它。不过与UML结合最好的是用例驱动的、以体系结构为中心的、迭代的、增量的开发过程。

18.1.4　UML 的应用领域

UML 的目标是以面向对象图的方式来描述任何类型的系统，具有很宽的应用领域。其中最常用的是建立软件系统的模型，但它同样可以用于描述非软件领域的系统，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。总之，UML 是一个通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行建模。此外，UML 适用于系统开发过程中从需求规格描述到系统完成后测试的不同阶段。在需求分析阶段，可以用用例来捕获用户需求。通过用例建模，描述对系统感兴趣的外部角色及其对系统（用例）的功能要求。分析阶段主要关心问题域中的主要概念（如抽象、类和对象等）和机制，需要识别这些类及它们相互间的关系，并用 UML 类图来描述。为实现用例，类之间需要协作，这可以用 UML 动态模型来描述。在分析阶段，只对问题域的对象（现实世界的概念）建模，而不考虑定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。这些技术细节将在设计阶段引入，因此设计阶段为构造阶段提供更详细的规格说明。

编程（构造）是一个独立的阶段，其任务是用面向对象编程语言将来自设计阶段的类转换成实际的代码。在用 UML 建立分析和设计模型时，应尽量避免考虑把模型转换成某种特定的编程语言。因为在早期阶段，模型仅仅是理解和分析系统结构的工具，过早考虑编码问题十分不利于建立简单正确的模型。

UML 模型还可作为测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试小组使用不同的 UML 图作为测试依据：单元测试使用类图和类规格说明；集成测试使用部件图和合作图；系统测试使用用例图来验证系统的行为；验收测试由用户进行，以验证系统测试的结果是否满足在分析阶段确定的需求。

总之，标准建模语言（UML）适用于以面向对象技术描述任何类型的系统，而且适用于系统开发的不同阶段，从需求规格描述直至系统完成后的测试和维护。

18.2　用例图

用例是什么呢？Ivar Jacobson 是这样描述的:“用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例定义一组用例实例。”

首先，从定义中得知用例是由一组用例实例组成的，用例实例也就是常说的“使用场景”，就是用户使用系统的一个实际的、特定的场景。其次，我们可以知道，用例应该给参与者带来可见的价值，这一点很关键。最后，我们得知，用例是在系统中的。

18.2.1 用例基本概念

用例模型描述的是外部执行者（Actor）所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。

在 UML 中，用例表示为一个椭圆。图 18-2 所示为一个个人图书管理系统的用例图。其中，“新增书籍信息”、“查询书籍信息”、“修改书籍信息”、“登记外借情况”、“查询外借情况”和“统计金额与册数”等都是用例的实例。

1．参与者（Actor）

参与者代表与系统接口的任何事物或人，它是指代表某一种特定功能的角色，因此参与者都是虚拟的概念。在 UML 中，用一个小人表示参与者。

图 18-2 中的“图书管理员”就是参与者。对于该系统来说，可能可以充当图书管理员角色的有多个人。由于他们对于系统而言均起着相同的作用，扮演相同的角色，因此只使用一个参与者表示。切记不要为每一个可能与系统交互的真人画出一个参与者。

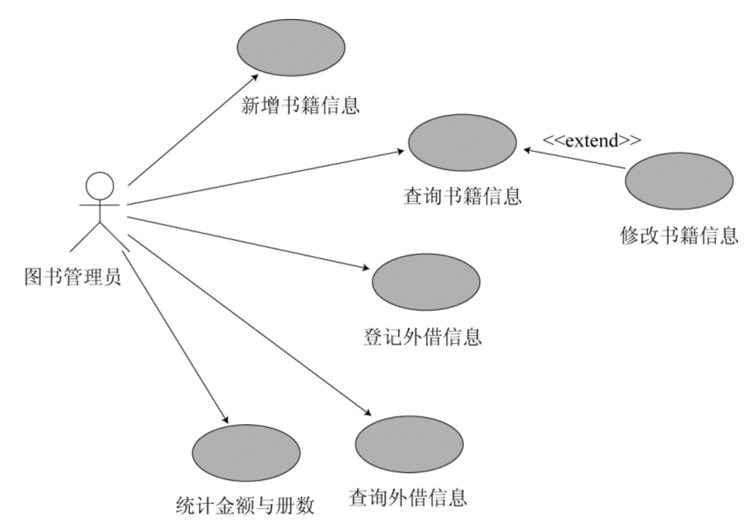


图 18-2　一个个人图书管理系统的用例图

2．用例（Use Case）

用例是对系统行为的动态描述，它可以促进设计人员、开发人员与用户的沟通，理解正确的需求，还可以划分系统与外部实体的界限，是系统设计的起点。在识别出参与者之后，可以使用下列问题识别用例：

- 每个参与者的任务是什么？
- 有参与者将要创建、存储、修改、删除或读取系统中的信息吗？
- 什么用例会创建、存储、修改、删除或读取这个信息？
- 参与者需要通知系统外部的突然变化吗？
- 需要通知参与者系统中正在发生的事情吗？
- 什么用例将支持和维护系统？
- 所有的功能需求都对应到用例中了吗？

- 系统需要何种输入/输出？输入从何处来？输出到何处？
- 当前运行系统的主要问题是什么？

3．包含和扩展（Include and Extend）

两个用例之间的关系主要可以概括为两种情况。一种是用于重用的包含关系，用构造型<<include>>表示；另一种是用于分离出不同的行为，用构造型<<extend>>表示。

1）*包含关系*

当可以从两个或两个以上的原始用例中提取公共行为，或者发现能够使用一个组件来实现某一个用例的部分功能是件很重要的事情时，应该使用包含关系来表示它们，如图 18-3 所示。

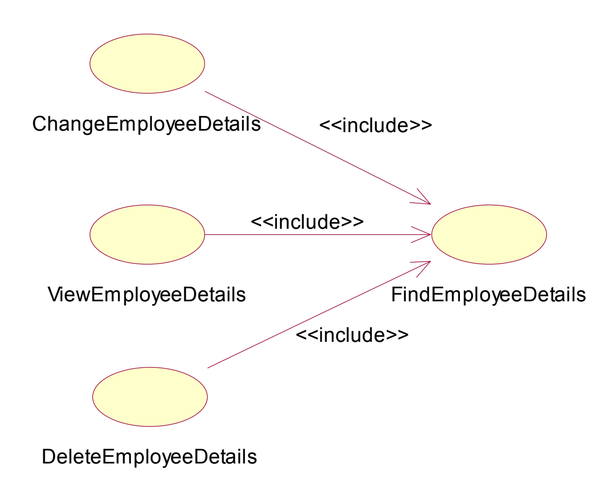


图 18-3　包含关系示例图

2）*扩展关系*

如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种事情，则可以将这个用例分为一个主用例和一个或多个辅用例，描述可能更加清晰，如图 18-4 所示。

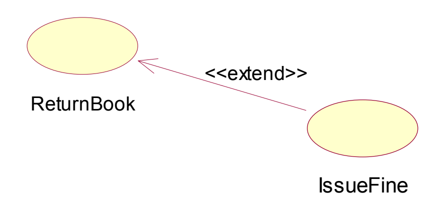


图 18-4　扩展关系示例图

18.2.2　构建用例模型

构建用例模型需要经历识别参与者、合并需求获得用例和细化用例描述 3 个阶段。

1．识别参与者

参与者（Actor）是同系统交互的所有事物，该角色不仅可以由人承担，还可以是其他系统、硬件设备，甚至是时钟。

- 其他系统：当系统需要与其他系统交互时，如在开发ATM柜员机系统时，银行后台系统就是一个参与者。
- 硬件设备：如果系统需要与硬件设备交互，如在开发IC卡门禁系统时，IC卡读写器就是一个参与者。
- 时钟：当系统需要定时触发时，时钟就是一个参与者，如在开发Foxmail中的“定时自动接收”功能时，就需要引入时钟作为参与者。

需要注意的是，参与者一定在系统之外，不是系统的一部分。通常可以通过如下问题来整理思路：

- 谁使用这个系统？
- 谁安装这个系统？
- 谁启动这个系统？
- 谁维护这个系统？
- 谁关闭这个系统？
- 哪些其他的系统使用这个系统？
- 谁从这个系统获取信息？
- 谁为这个系统提供信息？
- 是否有事情自动在预计的时间发生？

2．合并需求获得用例

将参与者都找到之后，接下来就是仔细地检查参与者，为每一个参与者确定用例。而其中的依据主要可以来源于已经获取的“特征表”。

1）将特征分配给相应的参与者

首先，要将这些捕获到的特征，分配给与其相关的参与者，以便可以针对每一个参与者进行工作，而无遗漏。而在本例中，所有的特征都是与一个参与者（即图书管理员）相关的。

2）进行合并操作

在合并之前，首先还要明确为什么要合并，知道了合并的目的，才能选择正确的合并操作。一个用例就是一个对参与者来说可见的价值结果，因此合并的根据就是使得其能够组合成为一个可见的价值结果。

合并后将产生用例，而用例的命名应该注意采用“动词（短语）+名词（短语）”的形式，而且最好能够对其进行编号，这也是实现跟踪管理的重要技巧，通过编号可以将用户的需求落实到特定的用例中去。

3）绘制成用例图

最后，将识别到的参与者，以及合并生成的用例通过用例图的形式整理出来。千万不要以为到此用例分析就结束了。这仅仅是一个好的开端，接下来的工作才是最为重要的一环，也是用例发挥作用的关键。

3．细化用例描述

接下来，针对图 18-2 中的一个用例“新增书籍信息”，说明如何细化用例描述。

1）搭框架

首先，根据特性表和前面的分析，先完成一个框架，内容如下。

1.用例名称：

新增书籍信息（UC01）

2.简要说明：

录入新购书籍信息，并自动存储建档

3.事件流：

3.1 基本事件流

3.2 扩展事件流

4.非功能需求

5.前置条件

用户进入图书管理系统

6.后置条件

完成新书信息的存储建档

7.扩展点

无

8.优先级

最高（满意度 5，不满意度 5）

每个部分写作时的注意点如下。

- 用例名称：应该与用例图相符，并写上其相应的编号。
- 简要说明：对该用例对参与者所传递的价值结果进行描述，应注意语言简要，使用用户能够阅读的自然语言。
- 前置条件：是执行用例之前必须存在的系统状态，这部分内容如果现在不容易确定可以在后面再细化。
- 后置条件：用例执行完毕系统可能处于的一组状态，这部分内容如果现在不容易确定也可以在后面再细化。
- 扩展点：如果包括扩展或包含用例，则写出扩展或包含用例名，并说明在什么情况下使用。而在本例中，用例图里没有相应的内容，因此可以直接写“无”。如果有，则应该在编写事件流的同时进行编写。
- 优先级：说明用户对该用例的期望值，可以为今后开发时制定先后顺序。可以采用满意度/不满意度指标进行说明，其中满意度的值为0～5，是指如果实现该功能，用户的满意程度；而不满意度的值也为0～5，是指如果不实现该功能，用户的不满意程度。

对于任何一个用例，在分析阶段都应该将其框架用例描述建立起来。

2）填血肉

在这个阶段的主要工作就是将事件流进行细化。在实际的开发工作中，要不要对一个用例进行细化、细化到什么程度主要根据项目迭代的计划来决定。

……

3.事件流：

3.1 基本事件流

1）图书管理员向系统发出“新增书籍信息”请求

2）系统要求图书管理员选择要新增的书籍是计算机类还是非计算机类

3）图书管理员做出选择后，显示相应界面，让图书管理员输入信息，并自动根据书号规则生成书号

4）图书管理员输入书籍的相关信息，包括：书名、作者、出版社、ISBN号、开本、页数、定价、是否有CDROM

5）系统确认输入的信息中书名未有重名

6）系统将所输入的信息存储建档

3.2 扩展事件流

a）如果输入的书名有重名现象，则显示出重名的书籍，并要求图书管理员选择修改书名或取消输入

a1）图书管理员选择取消输入，则结束用例，不做存储建档工作

a2）图书管理员选择修改书名后，转到5）

4.非功能需求

无特殊要求

……

在编写事件流的时候，应该注意如下几点。

- 使用简单的语法：主语明确，语义易于理解。
- 明确写出“谁控制球”：也就是在事件流描述中，让读者直观地了解是参与者在控制还是系统在控制。
- 从俯视的角度来编写：指出参与者的动作，以及系统的响应，也就是从第三者的角度来写。
- 显示过程向前推移：也就是每一步都有前进的感觉（例如，用户按下【Tab】键作为一个事件就是不合适的）。
- 显示参与者的意图而非动作（光有动作，让人不容易直接从事件流中理解用例）。
- 包括“合理的活动集”（带数据的请求、系统确认、更改内部、返回结果）。
- 用“确认”而非“检查是否”，例如，“系统确认所输入的信息中书名未有重名”。
- 可选择地提及时间限制。

另外，事件流的编写过程也是可以分阶段、迭代进行的，对于优先级高的用例花更多的时间，更加地细化；对优先级低的用例可以先简略地将主要事件流描述清楚再留到以后。

另外，对于一些较为复杂的事件流，可以在用例描述中引用顺序图、状态图、协作图等手段进行描述。

而在非功能需求小节中，主要对该用例所涉及的非功能性需求进行描述。由于其通常很难在事件流中进行表述，因此单列为一小节进行阐述。这些需求通过包括法律法规、应用程序标准、质量属性（可用性、可靠性、性能、支持性等）、兼容性、可移植性，以及设计约束等方面的需求。在这些需求的描述方面，一定要注意使其可度量、可验证，否则就容易流于形式，形同摆设。

3）补缺漏

在填血肉阶段要注意加强与用户的沟通，写完后需要与客户进行验证，然后不断地进行补缺漏，以保证用例描述完整、清晰、正确。

18.2.3　用例的粒度

用例作为一种有效的需求分析技术，近几年来被软件开发业界广泛采用和认同。虽然用例的形式比较简单，规则也不复杂，但正是由于这种自由性，要得心应手地灵活应用和发挥并不是一件很容易的事。其中最大的一个不容易把握的地方，就是用例的粒度，也就是多大才算是一个好的用例。

1．思辨“四轮马车”

在前面，我们通过合并特征获得了用例，在那里就留下了一个疑问。这个疑问其实就与用例的粒度相关。那就是笔者合并生成的用例中包括“新增书籍信息”、“修改书籍信息”和“查询书籍信息”，这 3 个刚好是犯了一个大名鼎鼎的错误——“四轮马车”！在新增、修改、查询、删除 4 个操作中，就引入了 3 个，很多大师都建议将其归结为一个——“管理书籍信息”。

那么，笔者又为什么要犯这个明知故犯的错误呢？其实，在大量的应用中都会涉及新增、修改、查询、删除的动作，因此如果在分析时把这些内容全都整理为一个用例，就会使得用例过多，复杂度太大，模型不够抽象。其实在具体的处理中，还是会将其作为子用例看待，用扩展的方式描述出来。而在本例中，系统相对简单，这几个功能将其独立出来并没有什么影响，而且这几个功能属于系统的重要核心功能，因此笔者认为这样处理并无不妥。当然这么说，并不是说“四轮马车”错误的总结不对，“四轮马车”的本意应该是指对非核心实体无须过度展开，如图书馆管理系统中的“管理会员信息”功能就不应该过度展开；另一方面，如果系统较大，也会使得用例的数量过多，大大提高了复杂度。

其实，从中我想表达出来的一种观点就是，用例的粒度其实是一个“度”的问题。而根据中国传统的中庸之道，度无绝对，也就是说，找不到一个绝对值来说明到底什么程度是对的，什么程度是错的。因此，大家不要为此所困，而是应该根据自己的需要来决定。不过，其中有一件很重要的事情，那就是不管用例的粒度大还是小，都需要符合“可见的价值结果”这一原则，否则就将违背了用例的思想，无法获得用例所带来的益处。

例如，“财务管理”，故意为了符合用例的命名规则，而改成类似“管理财务信息”的名称。作为一个用例，其实这是违背了用例的思想，因为它无法符合“可见的价值结果”的原则。它太大了，这样使使用用例的人还在用“功能分解”的思路理解系统。

再如，“输入支付信息”作为一个用例，认真一分析，就会发现它只是一个步骤，并不能够传达“可见的价值结果”。它太小了，这是一个过度使用用例的例子。

2．如何整理用例的层次

在实践中，经常看到实践者忍不住地将用例分成几个层次，先找到一些像“财务管理”这样的所谓的大用例，然后在后面用 include 或 extend 关系引入所谓的小用例，建立所谓的层次结构。其实这样的做法并不正确，应该通过包来表现用例层次。如果用例太多，就应该归类整理到一个个包里。下面就针对本例进行整理，当然该系统其实无须进行这一步，这里只是帮助大家理解。

- 书籍管理：包括“新增书籍信息”、“修改书籍信息”和“查询书籍信息”。
- 外借管理：包括“登记外借信息”和“查询外借信息”。
- 数据统计：包括“统计金额和册数”。

如果使用 Rational Rose 绘制模型，可以先画 3 个包，然后在 3 个包中分别绘制出相应的子图即可。如果使用其他工具，如 Visio、纸和笔，那么可以在用例图上将属于一个包的用例框在一起，并在框上写上包的名字即可。

18.3 类图和对象图

在面向对象建模技术中，我们将客观世界的实体映射为对象，并归纳成一个个类。类（Class）、对象（Object）和它们之间的关联是面向对象技术中最基本的元素。对于一个想要描述的系统，其类模型和对象模型揭示了系统的结构。

18.3.1 类与类图的基本概念

在 UML 中，类和对象模型分别由类图和对象图表示。类图技术是 OO 方法的核心。图 18-5 所示为一个小型图书管理系统的类图。

1．类和对象

对象（Object）与我们对客观世界的理解相关。我们通常用对象描述客观世界中某个具体的实体。所谓类（Class）是对一类具有相同特征的对象的描述。而对象是类的实例（Instance）。在 UML 中，类的可视化表示为一个划分成 3 个格子的长方形（下面两个格子可省略）。在图 18-5 中，“书籍”、“借阅记录”等都是一个类。

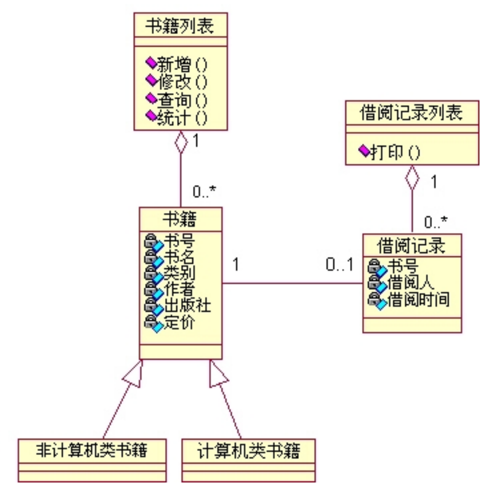


图 18-5　一个小型图书管理系统的类图

1）类的获取和命名

最顶部的格子包含类的名字。类的命名应尽量用应用领域中的术语，应明确、无歧义，以利于开发人员与用户之间的相互理解和交流。

2）类的属性

中间的格子包含类的属性，用以描述该类对象的共同特点。该项可省略。在图 18-5 中“书籍”类有“书名”、“书号”等特性。UML 规定类的属性的语法如下：

```
可见性  属性名：类型 = 默认值  {约束特性}
```

可见性包括 Public、Private 和 Protected，分别用“+”、“−”、“#”号表示。

类型表示该属性的种类，它可以是基本数据类型，如整数、实数、布尔型等，也可以是用户自定义的类型。一般它由所涉及的程序设计语言确定。

约束特性则是用户对该属性性质一个约束的说明。例如“{只读}”说明该属性是只读属性。

3）类的操作（Operation）

该项可省略。操作用于修改、检索类的属性或执行某些动作。操作通常也称为功能，但它们被约束在类的内部，只能作用到该类的对象上。操作名、返回类型和参数表组成操作界面。UML 规定操作的语法如下：

```
可见性：操作名（参数表）：返回类型 {约束特性}
```

类图描述了类和类之间的静态关系。定义了类之后，就可以定义类之间的各种关系。

2．类之间的关系

在建立抽象模型时，很少有类会单独存在，大多数类都会以某种方式彼此协作，因此还需要描述这些类之间的关系。关系是事物间的连接，在面向对象建模中，有 4 个很重要的关系。

1）依赖关系

有两个元素 X、Y，如果修改元素 X 的定义可能会引起对另一个元素 Y 的定义的修改，则称元素 Y 依赖（Dependency）于元素 X。在 UML 中，使用带箭头的虚线表示依赖关系，如图 18-6 所示。

图 18-6　依赖关系的图示

在类中，依赖由各种原因引起，例如，一个类向另一个类发消息；一个类是另一个类的数据成员；一个类是另一个类的某个操作参数。如果一个类的界面改变，它发出的任何消息可能不再合法。

2）泛化关系

泛化关系描述了一般事物与该事物中的特殊种类之间的关系，也就是父类与子类之间的关系。继承关系是泛化关系的反关系，也就是说子类是从父类中继承的，而父类则是子类的泛化。在 UML 中，使用带空心箭头的实线表示，箭头指向父类，如图 18-7 所示。

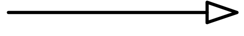


图 18-7　泛化关系的图示

在 UML 中，对泛化关系有如下 3 个要求：

- 子类应与父类完全一致，父类所具有的关联、属性和操作，子元素都应具有。
- 子类中除与父类一致的信息外，还包括额外的信息。
- 可以使用子父类实例的地方，也可以使用子类实例。

在如图 18-5 所示的例子中，“书籍”与“非计算机类书籍”之间就是泛化关系。

3）关联关系

关联（Association）表示两个类之间存在某种语义上的联系。例如，一个人为一家公司工作，一家公司有许多办公室。我们就认为人和公司、公司和办公室之间存在某种语义上的联系。

关联关系提供了通信的路径，它是所有关系中最通用、语义最弱的。在 UML 中，使用一条实线来表示关联关系。

- 聚合关系：又称为聚集关系，聚合（Aggregation）是一种特殊形式的关联。聚合表示类之间的关系是整体与部分的关系。例如，一辆轿车包含4个车轮、一个方向盘、一个发动机和一个底盘，就是聚合的一个例子。在UML中，使用一个带空心菱形的实线表示，空心菱形指向的是代表“整体”的类，如图18-8所示。



图 18-8　聚合关系的图示

- 组合关系：如果聚合关系中表示“部分”的类的存在，与表示“整体”的类有着紧密的关系，例如“公司”与“部门”之间的关系，那么就应该使用“组合”关系来表示。在UML中，使用带有实心菱形的实线表示。

4）实现关系

实现关系是用来规定接口和实现接口的类或组件之间的关系。接口是操作的集合，这些操作用于规定类或组件的服务。在 UML 中，使用一个带空心箭头的虚线表示，如图 18-9 所示。

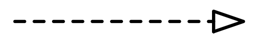


图 18-9　实现关系的图示

3．多重性问题

重复度（Multiplicity）又称为多重性，多重性表示为一个整数范围 *n..m*，整数 *n* 定义所连接的最少对象的数目，而 *m* 则为最多对象数（当不知道确切的最大数时，最大数用*号表示）。最常见的多重性有：0..1；0..*；1..1；1..*；*。

多重性用来说明关联的两个类之间的数量关系，例如：

- 书与借书记录之间的关系就应该是1对0..1的关系，也就是一本书可以有0个或1个借书记录。

- 经理与员工之间的关系则应为1对0..*的关系，也就是一个经理可以领导0个或多个员工。
- 学生与选修课程之间的关系就可以表示为0..*对1..*的关系，也就是一个学生可以选择一门或多门课程，而一门课程有0个或多个学生选修。

4．类图

对于软件系统，类模型和对象模型类图（Class Diagram）描述类和类之间的静态关系。与数据模型不同，它不仅显示了信息的结构，同时还描述了系统的行为。类图是定义其他图的基础。

5．对象图

UML 中对象图与类图具有相同的表示形式。对象图可以看作类图的一个实例。对象是类的实例；对象之间的链（Link）是类之间的关联的实例。对象与类的图形表示相似，均为划分成两个格子的长方形（下面的格子可省略）。上面的格子是对象名，对象名下有下画线；下面的格子记录属性值。链的图形表示与关联相似。对象图常用于表示复杂的类图的一个实例。

18.3.2　构建概念模型

在开发初始，我们需要通过类图来构建一个概念模型。那么什么是概念模型呢？“问题域”是指一个包含现实世界事物与概念的领域，这些事物和概念与所设计的系统要解决的问题有关。而建立概念模型，又称为问题域建模、域建模，也就是找到代表那些事物与概念的“对象”。

建立概念模型通常需经过如下几个步骤。

1．发现类

发现类的方法有很多种，其中最广泛应用的莫过于“名词动词法”，下面就采用该方法开始问题域建模的第一步。

注：名词动词法的主要规则是从名词与名词短语中提取对象与属性；从动词与动词短语中提取操作与关联；而所有格短语通常表明名词应该是属性而不是对象。

1）找到备选类

首先，可以逐字逐句地阅读上面那段需求描述，并将其中的所有名词及名词短语列出来。

2）决定候选类

很显然，并不是每一个备选类都是合适的候选类。有些名词对于要开发的系统来说无关紧要，甚至是系统之外的；而有些名词表述的概念则相对较小，适合于某个候选类的属性。因此，需要对备选类进行一番筛选，将这些不合适的类排除掉。

2．确定类之间的关联

通过上面的工作，从需求描述中找到与问题域紧密相关的类，接下来首要的任务就是理清类之间的层次关系。结合如图 18-5 所示的类模型来说明其分析过程。

对于这 6 个基本的类，可以发现“计算机类书籍”、“非计算机类书籍”与“书籍”之间是继承关系；而“书籍列表”则由多个“书籍”组成，“借阅记录列表”由多条“借阅记录”组成。另外，还可以发现“借阅记录”与“书籍”关联，离开“书籍”，“借阅记录”将失去意义。

为了反映和记录这些类之间的关联关系，就可以使用 UML 中的类图将其记录下来，如图 18-10 所示。

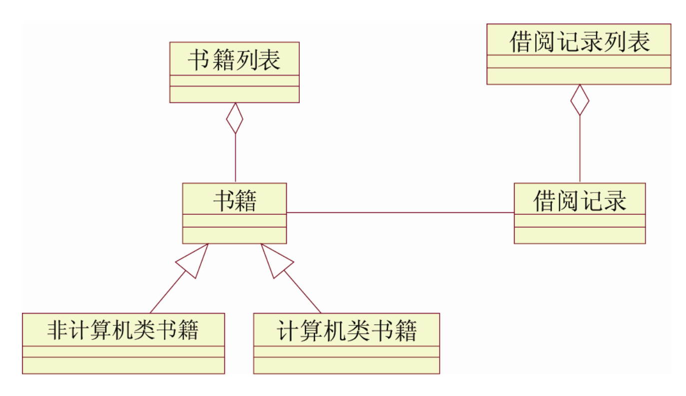


图 18-10　域建模中间过程

但是，图 18-10 无法将关联关系的细节信息传递出来。例如，一本书可以有几条借阅记录？书籍列表指的是多少本书籍？这些问题需要进一步分析，并修改上面所列出的类图。

- 系统应用于个人藏书管理，每本书都是唯一的，没有副本。因此其要么被借出去，要么未被借出。因此对于每一本书籍来说，要么没有借阅记录，要么也只有一条借阅记录。
- 所有的书籍组成书籍列表，借阅记录列表也是由所有的借阅记录组成的。

通过分析，可以将得到的信息补充到类图上，就可以得到如图 18-11 所示的关系图。

这样，我们就对所有问题域中的各个类之间的层次结构关系、协作关系有了一个完整的了解与认识。而对于较大的系统而言，还可以在此基础上对一些关联度大的部分类合成一个包，以便更好地抽象系统。

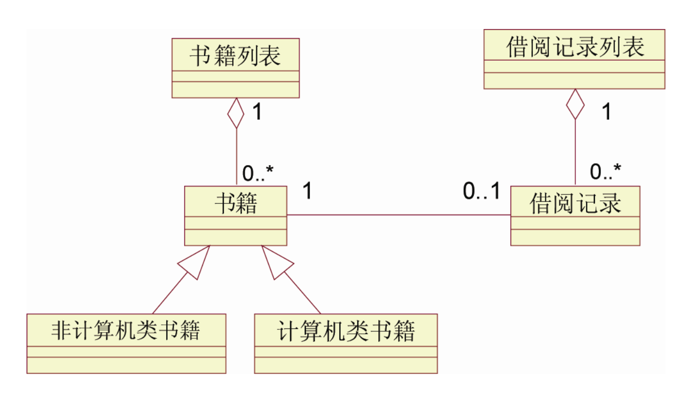


图 18-11　加上关联描述的概念模型

例如，在本例中可以将“书籍列表”、“书籍”、“非计算机类书籍”和“计算机类书籍”合成一个包，而将“借阅记录”与“借阅记录列表”合成一个包。不过本例比较简单，类也相对较少，因此无须进行分解。

3．为类添加职责

当找到了反映问题域本质的主要概念类，而且在理清它们之间的协作关系之后，就可

以为这些类添加其相应的职责。什么是类的职责呢？它包括如下两个主要内容：

- 类所维护的知识。
- 类能够执行的行为。

相信大家从上面的两句中，马上会想到类的成员变量（也称为属性）和成员方法吧！是的，成员变量就是其所维护的知识，成员方法就是其能够执行的行为。

在本阶段，需要根据需求描述的内容，以及与客户的简单沟通将主要类的主要成员变量和成员方法标识出来，以便更好地理解问题域。

- 书籍类：从需求描述中，已经找到了描述书籍的几个关键成员变量，即书号、书名、类别、作者、出版社；同时从统计的需要中，可以得知“定价”也是一个关键的成员变量。
- 书籍列表类：书籍列表就是全部的藏书列表，对于该类而言，其主要的成员方法是新增、修改、查询（按关键字查询）、统计（按特定时限统计册数与金额）。
- 借阅记录类：而针对“借阅记录”这个类，其关键的成员变量也一目了然，即书号、借阅人（朋友）、借阅时间。
- 借阅记录列表类：这也就是所有的借阅记录，对于该类而言其主要的职责就是打印借阅情况。

通过上述分析，可以使读者对这些概念类的了解更加深入，可以重新修改类图，将这些信息加入原先的模型，就得到了如图 18-5 所示的模型。

18.3.3 类模型的发展

在开发的初始阶段，对问题域进行建模，然后获得一个概念模型，但这个模型是远不足以指导开发的。因此，将根据用例分析的结果，进行更细化的设计，引入遗漏的类，加上一些与程序设计相关的控制类和边界类。

然后再逐步引入如 JDBC、MFC、Swing 等基础类，并结合设计模式、重构思想进行调整，最终发展成为一个反映代码结构的、详尽的类模型。

18.4 状态图

状态图（State Diagram）用来描述一个特定对象的所有可能状态及其引起状态转移的事件。大多数面向对象技术都用状态图表示单个对象在其生命周期中的行为。一个状态图包括一系列的状态及状态之间的转移。

图 18-12 所示为一个数码冲印店的订单状态图示例。

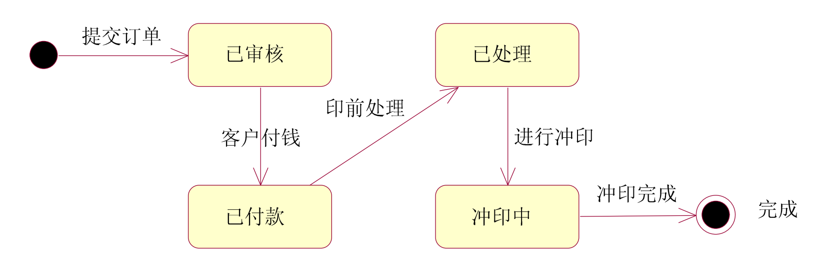


图 18-12　一个数码冲印店的订单状态图示例

如图 18-12 所示，状态图包括如下几个部分。

- 状态：又称为中间状态，用圆角矩形框表示。
- 初始状态：又称为初态，用一个黑色的实心圆圈表示，在一张状态图中只能够有一个初始状态。
- 结束状态：又称为终态，在黑色的实心圆圈外面套上一个空间圆，在一张状态图中可能有多个结束状态。
- 状态转移：用箭头说明状态的转移情况，并用文字说明引发这个状态变化的相应事件是什么。

一个状态也可能被细分为多个子状态，如果将这些子状态都描绘出来，那么这个状态就是复合状态。

状态图适合用于表述在不同用例之间的对象行为，但并不适合于表述包括若干协作的对象行为。通常不需要对系统中的每一个类绘制相应的状态图，而通常会在业务流程、控制对象、用户界面的设计方面使用状态图。

18.5 活动图

活动图的应用非常广泛，它既可用来描述操作（类的方法）的行为，也可以描述用例和对象内部的工作过程。活动图是由状态图变化而来的，它们各自用于不同的目的。活动图依据对象状态的变化来捕获动作（将要执行的工作或活动）与动作的结果。活动图中一个活动结束后将立即进入下一个活动（在状态图中状态的变迁可能需要事件的触发）。

1. 基本活动图

图 18-13 所示为一个基本活动图的示例。

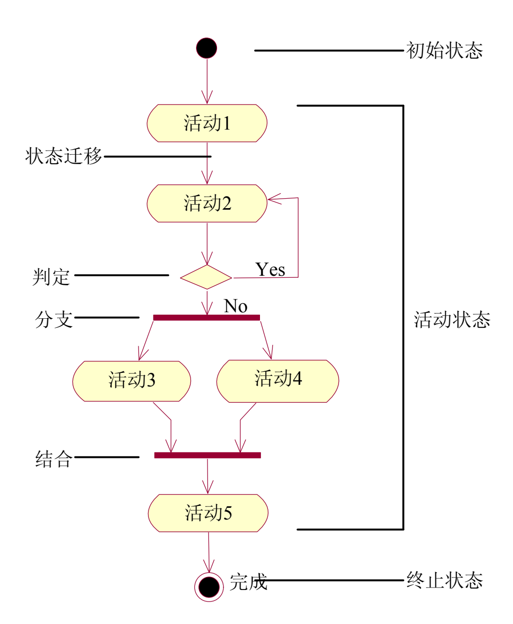


图 18-13　一个基本活动图的示例

如图 18-13 所示，活动图与状态图类似，包括初始状态、终止状态，以及中间的活动状态，每个活动之间，也就是一种状态的变迁。在活动图中，还引入了如下几个概念。

- 判定：说明基于某些表达式的选择性路径，在UML中使用菱形表示。
- 分叉与结合：由于活动图建模时经常会遇到并发流，因此在UML中引入了如图18-13所示的粗线来表示分叉和结合。

2．带泳道的活动图

在前面说明的基本活动图中，虽然能够描述系统发生了什么，但没有说明该项活动由谁来完成。而针对 OOP 而言，这就意味着活动图没有描述出各个活动由哪个类来完成。可以通过泳道来解决这一问题。它将活动图的逻辑描述与顺序图、协作图的责任描述结合起来。下面我们就一起来看一个使用了泳道的例子，如图 18-14 所示。

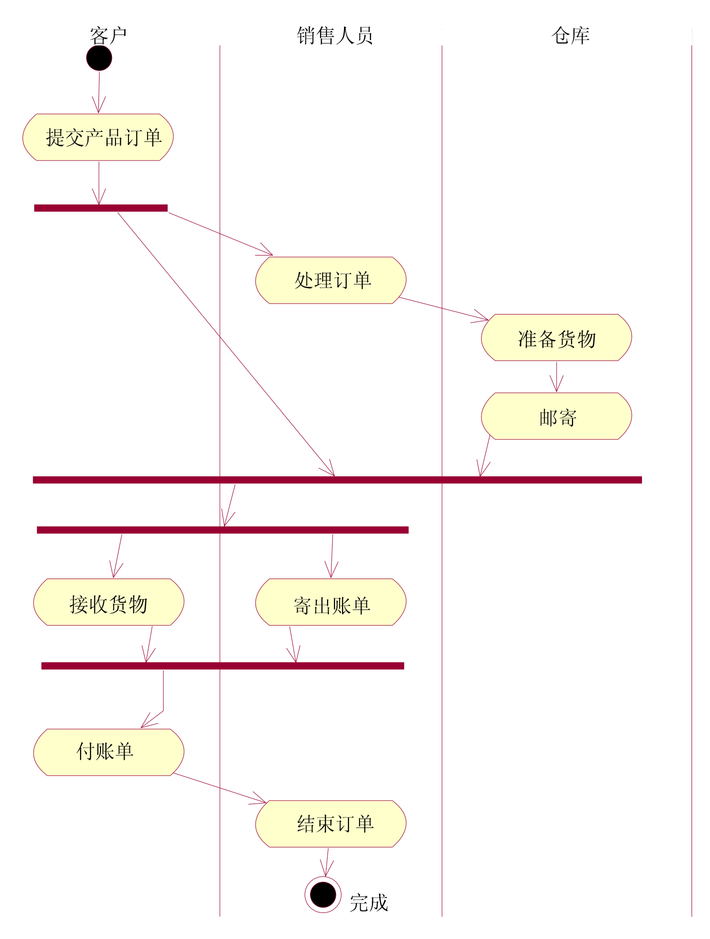


图 18-14　带泳道活动图示例

3．对象流

在活动图中可以出现对象。对象可以作为活动的输入或输出，对象与活动间的输入/输出关系用虚线箭头来表示。如果仅表示对象受到某一活动的影响，则可用不带箭头的虚线来连接对象与活动。

4．信号

在活动图中可以表示信号的发送与接收，分别用发送和接收标志来表示。发送和接收标志也可与对象相连，用于表示消息的发送者和接收者。

18.6　交互图

交互图（Interactive Diagram）是表示各组对象如何依某种行为进行协作的模型，通常可以使用一个交互图来表示和说明一个用例的行为。在 UML 中，包括两种不同形式的交互图，强调对象交互行为顺序的顺序图和强调对象协作的协作图，它们之间没有什么本质不同，只是排版不尽相同而已。

18.6.1　顺序图

顺序图（Sequence Diagram）用来描述对象之间动态的交互关系，着重体现对象间消息传递的时间顺序。顺序图允许直观地表示出对象的生存期，在生存期内，对象可以对输入消息做出响应，并且可以发送信息。

如图 18-15 所示，顺序图存在两个轴。水平轴表示不同的对象，即图中的 Client、Factory、Product 等；而垂直轴表示时间，表示对象及类的生命周期。

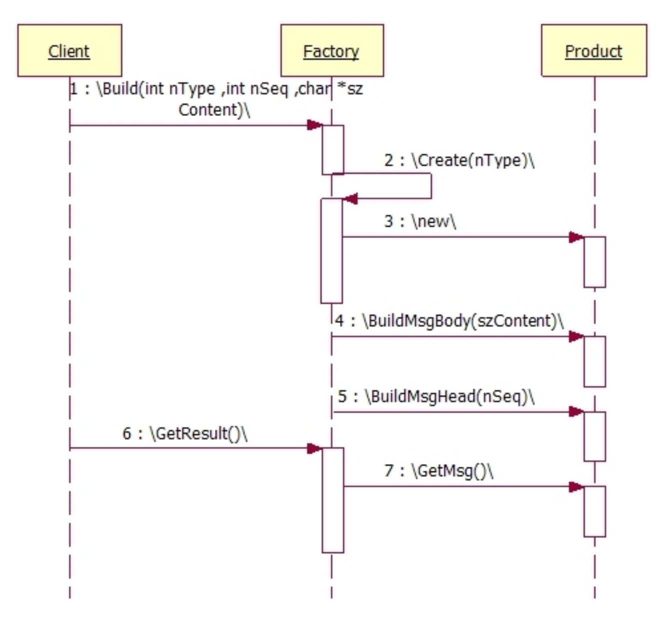


图 18-15　顺序图示例

对象间的通信通过在对象的生命线间画消息来表示。消息的箭头指明消息的类型。顺序图中的消息可以是信号、操作调用或类似于 C++中的 RPC（Remote Procedure Calls）和 Java 中的 RMI（Remote Method Invocation）。当收到消息时，接收对象立即开始执行活动，即对象被激活了。通过在对象生命线上显示一个细长矩形框来表示激活。

消息可以用消息名及参数来标识，消息也可带有顺序号。消息还可带有条件表达式，表示分支或决定是否发送消息。如果用于表示分支，则每个分支是相互排斥的，即在某一时刻仅可发送分支中的一个消息。

18.6.2　协作图（通信图）

通信图（Communication Diagram），该图在 UML1.x 中被称为协作图。用于描述相互合作的对象间的交互关系和链接关系。虽然顺序图和协作图都用来描述对象间的交互关系，但侧重点不一样。顺序图着重体现交互的时间顺序，协作图则着重体现交互对象间的静态

链接关系。图 18-16 就是与图 18-15 相对应的协作图，可以从图 18-16 中很明显地发现它与顺序图之间的异同点。

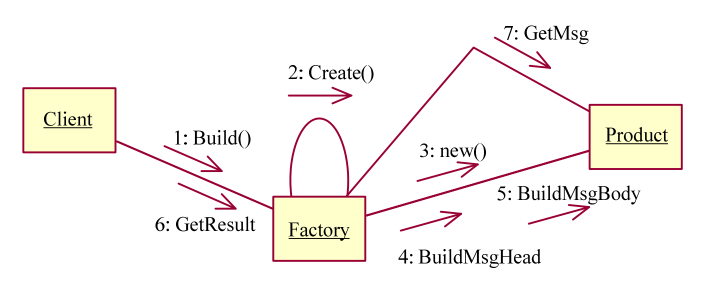


图 18-16　协作图示例

18.7　构件图

构件图是面向对象系统的物理方面进行建模时要用的两种图之一。它可以有效地显示一组构件及它们之间的关系。构件图中通常包括构件、接口及各种关系。图 18-17 所示为构件图示例。

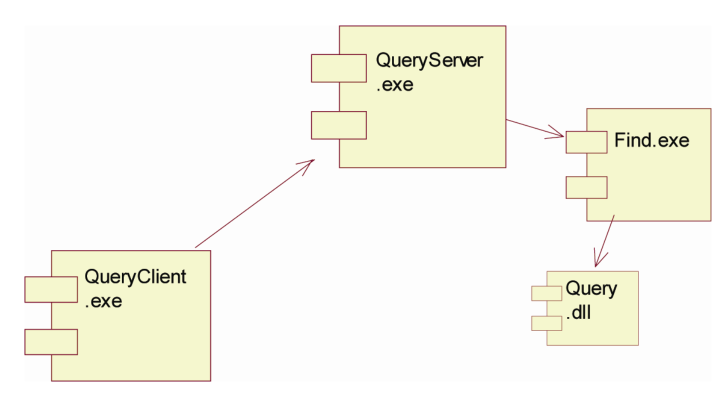


图 18-17　构件图示例

通常构件指的是源代码文件、二进制代码文件和可执行文件等。而构件图就是用来显示编译、链接或执行时构件之间的依赖关系。例如，在图 18-17 中，就是说明 QueryClient.exe 将通过调用 QueryServer.exe 来完成相应的功能，而 QueryServer.exe 则需要 Find.exe 的支持，Find.exe 在实现时调用了 Query.dll。

通常来说，可以使用构件图完成如下工作。

- 对源代码进行建模：这样可以清晰地表示出各个不同源程序文件之间的关系。
- 对可执行体的发布建模：如图18-17所示，将清晰地表示出各个可执行文件、DLL文件之间的关系。
- 对物理数据库建模：用来表示各种类型的数据库、表之间的关系。
- 对可调整的系统建模：例如对于应用了负载均衡、故障恢复等系统的建模。

在绘制构件图时，应该注意侧重于描述系统的静态实现视图的一个方面，图形不要过于简化，应该为构件图取一个直观的名称，在绘制时避免产生线的交叉。

18.8 包图

包图（Package Diagram）主要用于查看包之间的依赖性。虽然包图是在 UML2.0 中才新增的一种图，但在面向对象程序语言中包却早已普及。例如，C#中的命名空间、Java 中的 package 都是包。图 18-18 所示为包图示例，其中 Net 是与网络有关的类的集合。

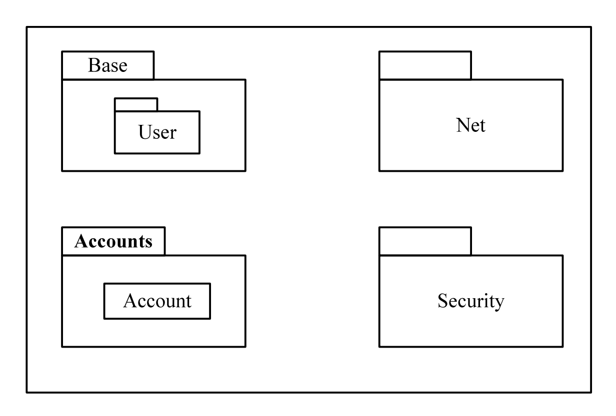


图 18-18　包图示例

包里面可以是类图、用例图及其他 UML 图（如 Accounts 包中包含 Account 类），也可以嵌套其他的包（如 Base 包中包含 User 包）。

1．包的依赖

图 18-18 中所示的包都是独立的包，但有时一个包的类需要用到另一个包的类，这就形成了包与包之间的依赖。依赖在图中用虚线箭头进行标识，若虚线箭头从 A 指向 B，则表示 A 依赖 B。如图 18-19 所示，Net 包与 Security 包都依赖 Base 包。

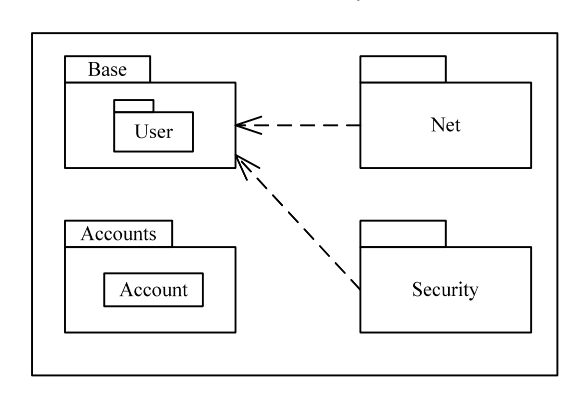


图 18-19　包的依赖示例

2．包的导入与访问

当一个包需要用到另一个包中的内容时，可以通过导入来实现。导入时可以导入整个包，也可以导入包中的部分元素。

导入的方式分为公共导入（import）和私有导入（access）。

- 公共导入：被导入的元素在将它们导入的包中具有public可见性。
- 私有导入：被导入的元素在将它们导入的包中具有private可见性。

如图 18-20 所示，Net 包以私有导入形式，导入了整个 Base 包，所以 Net 可以无须写全路径，直接使用 Base 包中的内容。而 Security 以公共导入形式，导入了 Accounts 包中的 Account 类。这属于部分导入，所以 Security 只能看到 Accounts 包中的 Account 类，看不到包中其他元素。

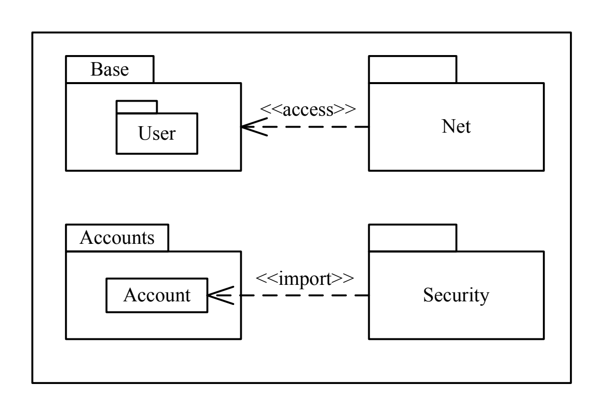


图 18-20　包的导入与访问示例

18.9　部署图

部署图，也称为实施图，它和构件图一样，是面向对象系统的物理方面建模的两种图之一。构件图用于说明构件之间的逻辑关系，而部署图则在此基础上更进一步，描述系统硬件的物理拓扑结构及在此结构上执行的软件。部署图可以显示计算结点的拓扑结构和通信路径、结点上运行的软件构件，常常用于帮助理解分布式系统。

图 18-21 所示为部署图示例，这样的图示可以使系统的安装、部署更为简单。

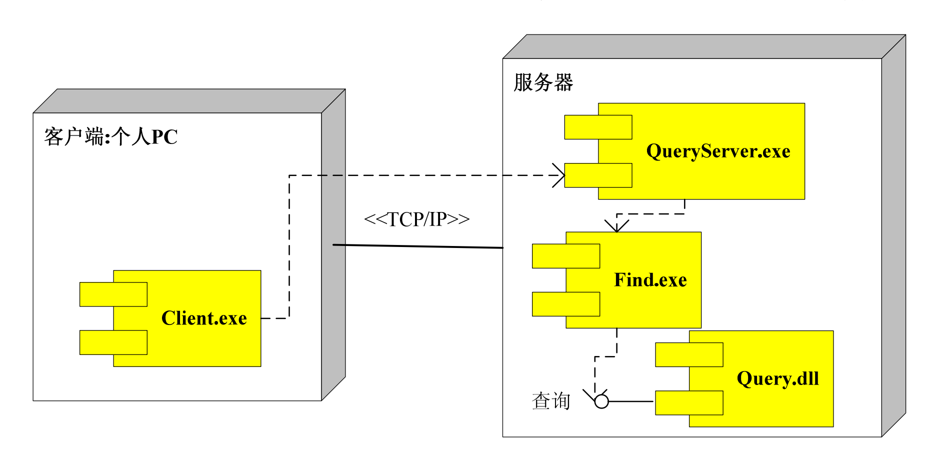


图 18-21　部署图示例

在部署图中，通常包括如下一些关键的组成部分。

1．结点和连接

结点（Node）代表一个物理设备及其上运行的软件系统，如一台 UNIX 主机、一个 PC 终端、一台打印机、一个传感器等。

如图 18-21 所示，“客户端：个人 PC”和“服务器”就是两个结点。在 UML 中，使用一个立方体表示一个结点，结点名放在左上角。结点之间的连线表示系统之间进行交互的通信路径，在 UML 中被称为连接。通信类型则放在连接旁边的“<<>>”之间，表示所用的通信协议或网络类型。

2．构件和接口

在部署图中，构件代表可执行的物理代码模块，如一个可执行程序。逻辑上它可以与类图中的包或类对应。在图 18-21 中，“服务器”结点中包含“QueryServer.exe”、“Find.exe”和“Query.dll”3 个构件。

在面向对象方法中，类和构件等元素并不是所有的属性和操作都对外可见。它们对外提供了可见操作和属性，称为类和构件的接口。界面可以表示为一头是小圆圈的直线。在图 18-21 中，“Query.dll”构件提供了一个“查询”接口。

第 19 章

数据库设计

从历年试题来看，在下午考试的软件设计试题中，也会出现数据库设计试题，主要考点包括数据的规范化、E-R 模型、数据库的逻辑设计与物理设计等内容。

19.1 数据的规范化

关系模式满足的确定约束条件称为范式，根据满足约束条件的级别不同，范式由低到高分为 1NF、2NF、3NF、BCNF、4NF、5NF 等。不同的级别范式性质不同。

关系模式的规范化就是把一个低一级的关系模式分解为高一级的关系模式的过程。

关系模式分解必须遵守如下两个准则。

- 无损连接性：信息不失真（不增减信息）。
- 函数依赖保持性：不破坏属性间存在的依赖关系。

规范化的基本思想是逐步消除不合适的函数依赖，使数据库中的各个关系模式达到某种程度的分离。规范化解决的主要是单个实体的质量问题，是对于问题域中原始数据展现的正规化处理。

规范化理论提供了判断关系模式优劣的理论标准，帮助我们预测模式可能出现的问题，是数据库逻辑设计的指南和工具，具体有如下两点：

- 用数据依赖的概念分析和表示各数据项之间的关系。
- 消除E-R图中的冗余联系。

19.1.1 函数依赖

函数依赖，通俗地说，就像自变量 x 确定之后，相应的函数值 $f(x)$也就唯一地确定了一样。函数依赖是衡量和调整数据规范化的最基础的理论依据。

比如记录职工信息的结构如下：

```
职工工号（EMP_NO）
职工姓名（EMP_NAME）
所在部门（DEPT）
```

EMP_NO 函数决定 EMP_NAME 和 DEPT，或者 EMP_NAME、DEPT 函数依赖于 EMP_NO，记为：EMP_NO→EMP_NAME，EMP_NO→DEPT。

19.1.2 码

关系 R<U,F>中的一个属性或一组属性 K，如果给定一个 K 则唯一决定 U 中的一个元组，也就是 U 完全函数依赖于 K，则称 K 为 R 的码。一个关系可能有多个码，选其中一个作为主码（主键）。

包含在任一码中的属性称为主属性，不包含在任何码中的属性称为非主属性。

关系 R 中的属性或属性组 X 不是 R 的码，但 X 是另一个关系模式的码，则称 X 是 R 的外码（外键）。

主码和外码是一种重要的表示关系间关联的手段。数据库设计中一个重要的任务就是要找到问题域中正确的关联关系，孤立的关系模式很难描述清楚业务逻辑。

19.1.3 1NF

第一范式（1NF）是最低的规范化要求。

如果关系 R 中所有属性的值域都是简单域，其元素（即属性）不可再分，是属性项而不是属性组，那么关系模式 R 是第一范式的，记作 R∈1NF。这一限制是关系的基本性质，所以任何关系都必须满足第一范式。第一范式是在实际数据库设计中必须首先达到的，通常称为数据元素的结构化。

职工工号	职工姓名	住　址
4110973	吕利英	陕西省西安市北大街 17 号[710001]

上表为非第一范式，分解如下。

职工工号	职工姓名	所在省	所在市	详细地址	邮　编
4110973	吕利英	陕西省	西安市	北大街 17 号	710001

这就满足了第一范式。经过处理后，就可以以省、市为条件进行查询和统计。

满足 1NF 的关系模式会有许多重复值，并且增加了修改其数据时引起疏漏的可能性。为了消除这种数据冗余和避免更新数据的遗漏，需要更加规范的第二范式（2NF）。

19.1.4 2NF

如果一个关系 R 属于 1NF，且所有的非主属性都完全地依赖于主属性，则称为第二范式，记作 R∈2NF。

为了说明问题，现举一个例子来说明。

有一个获得专业技术证书的人员情况登记表结构为：省份、姓名、证书名称、证书编号、核准项目、发证部门、发证日期、有效期。

这个结构符合 1NF，其中 “证书名称”和“证书编号”是主码，但是因为“发证部门”只完全依赖于“证书名称”，即只依赖于主关键字的一部分（即部分依赖），所以它不符合 2NF，这样首先存在数据冗余，因为证书种类可能不多。其次，在更改发证部门时，如果漏改了某一记录，存在数据不一致性。再次，如果获得某种证书的职工全部跳槽了，那么这个发证部门的信息就可能丢失了，即这种关系不允许存在某种证书没有获得者的情况。可以用分解的方法消除部分依赖的情况，而使关系达到 2NF 的标准。方法是从现有关系中分解出新的关系表，使每个表中所有的非关键字都完全依赖于各自的主关键字。可以将其分解成两个表，分别是：省份、姓名、证书名称、证书编号、核准项目、发证日期、有效

期，证书名称、发证部门。这样即完全符合 2NF。

19.1.5　3NF

如果一个关系 R 属于 2NF，且每个非主属性不传递依赖于主属性，这种关系是 3NF，记作 R∈3NF。

从 2NF 中消除传递依赖，就是 3NF。比如有一个表（职工姓名、工资级别、工资额），其中职工姓名是关键字，此关系符合 2NF，但是因为工资级别决定工资额，也就是说非主属性“工资额” 传递依赖于主属性“职工姓名”，它不符合 3NF，同样可以使用投影分解的办法将其分解成两个表：职工姓名、工资级别，工资级别、工资额。

19.1.6　BCNF

一般满足 3NF 的关系模式已能消除冗余和各种异常现象，能够获得比较满意的效果，但无论 2NF 还是 3NF 都没有涉及主属性间的函数依赖，所以有时仍会引起一些问题。由此引入 BC 范式（BCNF，Boyeet 和 Codd 提出）。通常认为 BCNF 是第三范式的改进。

BC 范式的定义：如果关系模式 R∈1NF，且 R 中每一个函数依赖关系中的决定因素都包含码，则 R 是满足 BC 范式的关系，记作 R∈BCNF。

当一个关系模式 R∈BCNF，则在函数依赖范畴中，就认为已彻底实现了分离，消除了插入、删除的异常。

综合 1NF、2NF 和 3NF、BCNF 的内涵可概括如下：

- 非主属性完全函数依赖于码（2NF的要求）。
- 非主属性不传递依赖于任何一个候选码（3NF的要求）。
- 主属性对不含它的码完全函数依赖（BCNF的要求）。
- 没有属性完全函数依赖于一组非主属性（BCNF的要求）。

19.1.7　逆规范化处理

规范化设计所带来的性能问题在实际应用中可能令人无法忍受。如果出现这种情况，就要进行逆规范化。逆规范化就是为了获得性能上的要求所进行的违反规范化规则的操作。由于逆规范化几乎必然导致冗余，占用更多的存储空间，因此它需要对性能和空间的平衡进行考虑，需要不断地尝试和评估过程。进行逆规范化有很多方法，不过大部分都与实际应用有关系，包括冗余属性、合并等，可以根据实际的应用进行选择，找到最有效的方法。

19.2　数据库设计概述

数据库设计涉及范围很广，要设计一个性能良好的数据库并非易事。从本质上讲，数据库设计的过程是将数据库系统与现实世界密切地、有机地、协调一致地结合起来的过程。数据库的设计质量与设计者的知识、经验和水平密切相关。作为数据库应用系统的重要组成部分，数据库设计的成败往往直接关系到整个应用系统的成败。

数据库的设计过程可分解为若干相互独立又相互依存的阶段，每一阶段采用不同的技术与工具，解决不同的问题，从而将问题局部化，减少了局部问题对整体设计的影响。目前，此方法已在数据库设计中得到广泛应用并获得较好的效果。

通常将数据库的设计分为需求分析、概念结构设计、逻辑结构设计和数据库物理设计4个阶段，如图19-1所示。

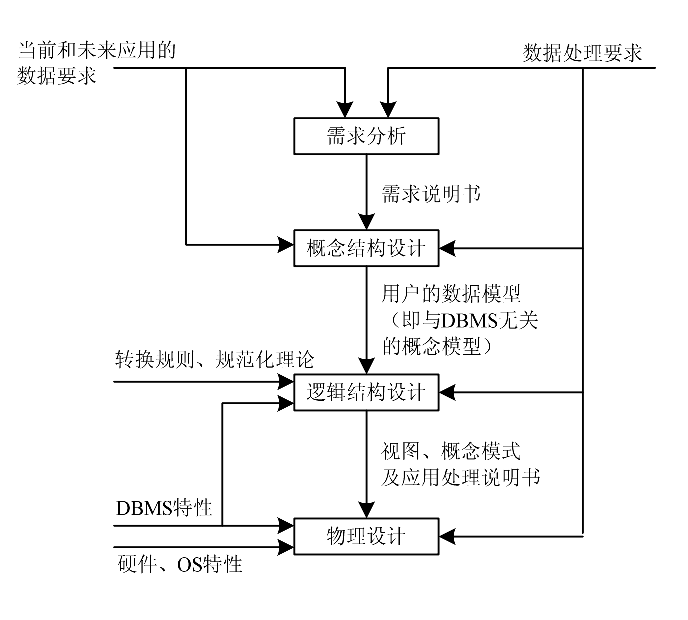


图19-1　数据库的设计步骤

1．需求分析

需求分析是指收集和分析用户对系统的信息需求和处理需求，得到设计系统所必需的需求信息，建立系统说明文档。其目标是通过调查研究，了解用户的数据要求和处理要求，并按一定格式整理形成需求说明书。需求说明书是需求分析阶段的成果，也是今后设计的依据，它包括数据库所涉及的数据、数据的特征、使用频率和数据量的估计，如数据名、属性及其类型、主关键字属性、保密要求、完整性约束条件、更改要求、使用频率、数据量估计等。这些关于数据的数据称为元数据（metadata）。在设计大型数据库时，这些数据通常由称为数据字典（Data Dictionary）的计算机软件（专用软件包或DBMS）来管理。用数据字典管理元数据有利于避免数据的重复或重名以保持数据的一致性及提供各种统计数据，因而有利于提高数据库设计的质量，同时可以减轻设计者的负担。

2．概念结构设计

概念结构设计是数据库设计的第二阶段，其目标是对需求说明书提供的所有数据和处理要求进行抽象与综合处理，按一定的方法构造反映用户环境的数据及其相互联系的概念模型，即用户的数据模型或企业数据模型。这种概念数据模型与DBMS无关，是面向现实世界的、极易为用户所理解的数据模型。为保证所设计的概念数据模型能正确、完全地反映用户（一个单位）的数据及其相互关系，便于进行所要求的各种处理，在本阶段设计中可吸收用户参与和评议设计。在进行概念结构设计时，可先设计各个应用的视图（View），即各个应用所看到的数据及其结构，然后再进行视图集成（View Integration），以形成一个单一的概念数据模型。这样形成的初步数据模型还要经过数据库设计者和用户的审查与修改，最后形成所需的概念数据模型。

3．逻辑结构设计

这一阶段的设计目标是把上一阶段得到的与DBMS无关的概念数据模型转换成等价的，并为某个特定的DBMS所接受的逻辑模型所表示的概念模式，同时将概念设计阶段得到的应用视图转换成外部模式，即特定DBMS下的应用视图。在转换过程中要进一步落实需求说明，并满足DBMS的各种限制。该阶段的结果是DBMS所提供的数据定义语言（DDL）写成的数据模式。逻辑设计的具体方法与DBMS的逻辑数据模型有关。逻辑模型应满足数据库存取、一致性及运行等各方面的用户需求。

4．数据库物理设计

物理设计阶段的任务是把逻辑设计阶段得到的满足用户需求的已确定的逻辑模型在物理上加以实现，其主要的内容是根据DBMS提供的各种手段，设计数据的存储形式和存取路径，如文件结构、索引的设计等，即设计数据库的内模式或存储模式。数据库的内模式对数据库的性能影响很大，应根据处理需求及DBMS、操作系统和硬件的性能进行精心设计。

实际上，数据库设计的基本过程与任何复杂系统开发一样，在每一阶段设计基本完成后，都要进行认真的检查，看看是否满足应用需求，是否符合前面已执行步骤的要求和满足后续步骤的需要，并分析设计结果的合理性。在每一步设计中，都可能发现前面步骤遗漏或处理不当之处，此时，往往需要返回去重新处理并修改设计和有关文档。所以，数据库设计过程通常是一个反复修改、反复设计的迭代过程。

19.3　需求分析

需求分析是数据库设计过程的第一步，是整个数据库设计的依据和基础。需求分析做得不好，会导致整个数据库设计重新返工。

19.3.1　需求分析的任务

需求分析的目标是通过对单位的信息需求及处理要求的调查分析得到设计数据库所必需的数据集及其相互联系，形成需求说明书，作为后面各设计阶段的基础。因此，这一阶段的任务是：

- 确认需求，明确设计目标。
- 分析和收集所需求的数据。
- 整理文档。

1．确认需求、明确设计目标

数据库设计的第一项工作就是通过实地调查研究，弄清现行系统的组织结构、功能划分、总体工作流程，明确用户总的需求目标；通过分析，确定相应的设计目标，即确定数据库应支持的应用功能和应用范围，明确哪些功能由计算机完成或准备让计算机完成，哪些环节由人工完成，以确定应用系统实现的功能。

要确定数据库应用领域，以有效地利用计算机设备及数据库系统的潜在能力，充分满足用户的应用要求，同时，还应当尽可能地考虑将来的应用需要，以提高数据库的应变能力，避免运行过程中对数据库做过多或较大的修改，以延长数据库的生命周期。

2．分析和收集所需求的数据

这是整个需求分析的核心任务。它包括分析和收集用户的信息需求、处理需求、完整性需求、安全性需求及对数据库设计过程有用的其他信息。

信息需求是指设计目标范围内涉及的所有实体、实体的属性及实体间的联系等数据对象，包括用户在数据处理中的输入/输出数据及这些数据间的联系。在收集中，要收集数据的名称、类型、长度、数据量、对数据的约束及数据间联系的类型等信息。

处理需求是指为了获得所需的信息而对数据加工处理的要求。它主要包括，处理方式是实时还是批处理，各种处理发生的频度、响应时间、优先级别，以及安全保密要求等。所要收集的其他信息还有企业在管理方式、经营方式等方面可能发生的变化等。

分析和收集数据的过程是数据库设计者对各类管理活动进行深入调查研究的过程，调查对象包括数据管理部门的负责人、各使用部门的负责人及操作员等各类管理人员，通过与各类管理人员相互交流，逐步取得对需求的一致认识。由于这种交流的双方知识背景不同，涉及不同层次、不同业务领域的管理人员，因而这一调查研究过程十分复杂和困难。为了减少因分析和收集数据的失误而导致整个系统的失败或延长研制周期，人们研究了许多用于需求分析的方法和技术，力求清楚地表达用户需求，系统地分析和收集需求数据。按方法的规范化程度可分为弱方法学（Week Methodologies）、强方法学（Strong Methodologies）和格式化方法（Formatted Approaches）。

3．整理文档

分析和收集得到的数据必须经过筛选整理，并按一定格式和顺序记载保存，经过审核成为正式的需求说明文档，即需求说明书。实际上，需求说明书是在需求分析的过程中逐渐整理形成的，是随着这一过程的不断深入而反复修改与完善的，是对系统需求分析的全面描述，由用户、领导和专家共同评审，是以后各设计阶段的主要依据。这一步的工作是进行全面的汇总、整理与系统化，以形成标准化的统一形式。

需求说明书作为应用部门的业务人员和数据库设计人员的“共同语言”，要求准确地表达用户需求，无二义性，可读性强，为数据库的概念设计、逻辑设计和物理设计提供全面、准确和详细的资料。

19.3.2　确定设计目标

数据库设计的首要任务是要确定应该支持的应用功能和应用领域。理想情况下，最好能覆盖一个企业的所有应用领域并尽可能地考虑到将来的应用需求。但由于财力、技术及管理基础等因素，这样做往往行不通，实际开发的数据库系统一般只包含基本的应用功能。因此首先要确定一个合理的需求目标，据此划定数据库设计的范围，即涉及的应用领域和部门。为此，可引用前面系统分析中的有关成果，如企业的职能机构与确定的目标有关的功能域等。

作为企业或部门，最好有自己的总体“信息计划”。计划范围可以很广，包括当前的和未来的信息系统战略、每个子系统的功能、各子系统间的相互依赖关系及数据分类情况。以此计划作为指南，就能很快地确定数据库的设计范围。如果企业或部门没有“信息计划”或计划没有足够的信息，开发人员就必须通过实地调查研究，弄清现行系统的组织结构、功能划分、总体工作流程、系统目标、存在困难与问题等，确定数据库的设计范围。其主要方法是根据最高决策机构的意图，了解数据库所涉及的部门和信息主题，向有关部门及

领导进行调查，确定数据库的服务范围。

确定数据库服务范围时一定要考虑将来需求的变动情况，这种变动可能来自将来业务的调整、经营策略的改变、政策环境的变化等。对于可能影响数据库的因素，就应把它纳入设计范围，使数据库适应这些变动。

19.3.3 数据收集与分析

确定数据库服务范围后就可着手进行系统需求信息的收集工作。信息收集所采取的方法主要有如下几种。

- 向各级用户发放问题调查表，以了解各部门所涉及的所有业务的工作名称、功能、目标结果等，形成工作总表。
- 查阅各有关部门现成的数据文档。这里文档包括表格、票据及报告、目前所使用的数据文件、旧系统的一切文档等。
- 与用户交谈，进一步明确每一业务的功能、处理过程、业务所需数据元素及各元素间的依赖关系、业务执行的规律、与其他业务的联系等。交谈对象必须具有代表性、普遍性，既可以是各业务的代表，也可以是各级管理决策者。交谈的方式可不拘一格。

所有信息收集工作不应以任何用户观点而应以数据观点来进行，这样可更方便、准确地进行数据分析整理。信息的收集整理工作包括如下几步：

①编辑整理出所有系统产生和使用的数据元素。

②定义所有业务中的事务及其所用到的数据。事务是由操作序列组成的、完成某一特定任务的、不可分割的工作单元。

③找出明显的或隐含的操作规则和政策。

④列出将来可能的变动情况及其对系统的影响。

为了高效地进行需求分析，必须采用好的工具和方法。结构化分析方法就是一种广泛应用的需求分析方法。它是以数据流图为主要工具、逐步求精地建立系统模型的一种系统分析方法。这种方法还采用了如数据字典、判定表、判定树等一些辅助工具，这些工具是配合使用的。

有关结构化分析方法及其数据流图与数据字典的有关介绍可参考相应章节的详细介绍，这里不再赘述。要说明的是，作为结构化分析方法的一个有力工具，其中的数据字典与DBMS中的数据字典在内容上有所不同，DBMS数据字典用来描述（或）定义数据库系统运行所涉及的各种对象，但在功能上它们是一致的。利用数据字典可更方便地管理软件开发过程或数据库系统，提高软件开发效率或数据库系统的运行效率。

下面对面向数据的方法进行介绍。

19.3.4 需求说明书

数据库设计遵循软件工程的原则和方法，其需求分析阶段的文档工作主要有如下几点。

1．整理调查材料

由于调查分析工作可能分成若干调查组同时进行，同一项业务有不同的调查对象，也可能分批分次进行调查，这样汇集的调查材料难免有不一致、不规范及粗糙、模糊等问题，

必须进行整理，使之成为正规、系统、清晰可读的资料。整理工作包括如下内容：

- 检查材料的完整性和系统性，即检查各项调查内容是否齐全，有关数据载体（单据、票证、表格等）是否收集齐全，应建的文档是否都建了，逻辑顺序是否合理，必要时加以追补和修改。
- 消除模糊点，修改错误，必要时可再次访问调查对象或请示主管领导。
- 消除命名冲突和冗余。命名冲突指各类数据的同名异义（homonym）及同义异名（synonym）问题。对于同名异义问题可通过鉴别同名数据元素的语义加以识别，然后加以消除。对于同义异名的识别则要困难一些，办法之一是对数据元素按语义进行分类，将语义相近的数据元素归在一起，以便缩小搜索范围。例如，按照实体名、日期、总计、单据等分类。类别分得越细，鉴别搜索范围就越小，然后再对小范围中不同名的数据元素的语义逐个鉴别其是否属同义。在此过程中也能识别出冗余数据，一旦发现应将其除去。

对已经识别的上述两类冲突，分别列表存档，留待以后进一步分析处理。

- 保持文档格式的一致性和规范化。
- 对所有原始材料及所做的文档进行编码。

2．用户审核确认

上述整理出来的调查分析材料仅仅表示了分析人员对单位的一种理解，这种理解只有和用户的想法相一致才有效，因此必须有一个统一认识的过程，这个过程就是用户审核确认的过程。用户审核确认的主要内容如下：

- 确认数据库的设计目标及范围是否合理。
- 验证每个业务功能流程的正确性及相关的数据对象是否齐全。
- 检查有关的事务规则是否正确完整。
- 数据的处理要求是否完全。
- 是否考虑到将来可能的变化或发展。

由于调查过程的复杂性，调查材料中不可避免地存在着误解与遗漏，如不及时加以解决，可能会有大的反复，所以用户的审核和确认至关重要，关系到数据库系统能否成功运行，必须给予足够的重视。

3．建立数据库设计的需求说明文档

用户对调查分析材料审核和确认后，就可着手建立数据库设计的需求说明文挡，即需求说明书。该需求说明书不是囊括全部调查分析材料，而只是选择对数据库设计直接有用的信息，一般包括实体类、联系类、数据的使用要求及冲突表等方面的内容。

（1）实体类：列出涉及的所有实体类并加以描述。

- 实体类名称及其语义说明，如图书这个实体类包括出版社发行的所有书籍。
- 可能的最大实例数，如图书总数为1万册。
- 描述实体类的数据元素，其格式如下。

数据元素名	简要描述	类型	域宽	存在概率	重复因子

其中，简要描述是对该数据元素的语义说明，存在概率表示该数据元素出现空值的概率，重复因子表示该元素值重复出现的最大次数，若其值为 1，表示不重复，可用作实体

的标识；如其值大于 1，表示可能重复出现，就不宜用作实体标识。

（2）联系类：对每个联系类要说明如下几点。

- 联系类名称、参与联系的实体及其他语义信息。
- 最大的实例数。

（3）事务处理说明：主要包括事务处理的如下内容。

- 过程描述。
- 执行者。
- I/O数据。
- 执行频率。
- 执行条件与后果。

（4）数据使用说明：对数据的使用要求应说明如下内容。

- 各种数据在相应事务中的使用类型，使用类型可为C（建立）、Q（查询）、I（插入）、M（修改）、D（删除）、N（不能操作）、ALL（Q、I、M、D）等。可用数据－处理关系矩阵表来表示。
- 用户组，即参与同一类事务活动的用户，可用事务活动名标识。
- 处理方式，分联机处理和批处理两类。联机处理如即席查询和更新等操作；批处理如各种例行报表的生成等。
- 数据使用的描述，说明数据在事务处理活动中被加工处理的过程。
- 数据的使用频度，按每日、每周使用数据的平均数和最高数表示，低于每周（月、日）一次者，标以不经常。

（5）冲突表：将已经识别的命名冲突用如图 19-2 所示的表格列表说明。

同名异义表

数据名称	语义说明	文档号

（a）

同义异名表

<table>
<tr><td>数据名称</td><td>语义说明</td><td>文档号</td></tr>
<tr><td></td><td></td><td rowspan="3"></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td><td rowspan="3"></td></tr>
<tr><td></td><td></td></tr>
<tr><td>……</td><td>……</td></tr>
</table>

（b）

图 19-2　冲突表

（6）运行需求说明：用户在数据库系统运行时对数据操作方面的要求说明。主要包括如下内容。

- 安全与保密性要求。
- 完整性要求。

- 响应时限要求。
- 后备与恢复要求。
- 系统扩展要求等。

19.4 概念结构设计

概念结构设计阶段所涉及的信息不依赖于任何实际实现时的环境，即计算机的硬件和软件系统。概念结构设计的目标是产生一个用户易于理解的，反映系统信息需求的整体数据库概念结构。概念结构设计的任务是，在需求分析中产生的需求说明书的基础上按照一定的方法抽象成满足应用需求的用户（单位）的信息结构，即通常所称的概念模型。概念结构的设计过程就是正确选择设计策略、设计方法和概念数据模型并加以实施的过程。

19.4.1 概念结构

概念模型是从现实世界到机器世界的一个过渡的中间层次，它有如下特点。

- 概念模型有丰富的语义表达能力，能表达用户的各种需求，是对现实世界的抽象和概括，它真实、充分地反映了现实世界中事务和事务之间的联系，能满足用户对数据的处理要求。
- 易于交流和理解。由于概念模型简洁、明晰、独立于机器，是数据库设计人员和用户之间的主要交流工具，因此可以用概念模型和不熟悉计算机的用户交换意见，使用户能积极参与数据库的设计工作，保证设计工作顺利进行。
- 概念模型易于更改。当应用环境和应用要求发生变化时，能方便地对概念模型修改和扩充，以反映这些变化。
- 概念模型很容易向关系、网状、层次等各种数据模型转换，易于导出与DBMS有关的逻辑模型。

概念数据模型的作用是：提供能够识别和理解系统要求的框架；为数据库提供一个说明性结构，作为设计数据库逻辑结构即逻辑模型的基础。

概念模型的描述工具应该能够体现概念模型的特点，如 E-R 模型。近年来，由于面向对象数据模型具有更丰富的语义、更强的描述能力而越来越受到人们的重视，不但出现了商品化的面向对象 DBMS，而且开始实际应用于概念模型的设计中，作为数据库概念设计的工具。Teory 等人提出的扩展的 E-R 模型（称为 E-E-R 模型）增加了类似面向对象数据模型中的普遍化和聚合等语义描述机制，为这种最为人们熟悉的数据模型注入了新的生机，为概念模型的描述增加了一种理想的选择。

概念结构的设计策略主要有自底向上、自顶向下、由里向外和混合策略。在具体实现设计目标时有两种极端的策略或方案，一是建立一个覆盖整个单位所有功能域的全局数据库，称为全局方案（Global Approach）或全局策略；另一种则是对每一个应用都建立一个单独的数据库，称为应用方案（Application Approach）或应用策略。

全局策略要求设计一个能够支持单位所有应用需求的单一数据库，此法在数据库应用早期曾受到广泛的推崇。但实践证明，对于一个较大的单位，采用此种策略设计数据库，会使设计任务十分复杂，开发周期过长，因而很难取得成功。应用策略则对每个应用单独设计一个数据库，优点是简化了分析工作且单库单用，运行效率较高，但也存在明显的缺点，会造成大量数据冗余、数据的不一致性、破坏整个单位中数据的完整性及难以实现数

据共享。因此，这一策略与传统的文件系统并无本质上的差别，从而与全局策略一样可能导致失败。

介于全局和应用两种策略之间的一个折中的策略是根据对事务活动的分析，先设计出一个单位的初步信息结构。该结构表示单位活动中涉及的主要实体、实体间的联系。然后根据对信息流的分析，按自然合理的分组原则将这些实体及实体间的联系分别组成若干便于管理的较小数据库，再进行每个数据库的详细设计。

折中策略在大多数情况下提供了一种最好的选择。需要注意的是，对初步信息结构中实体及实体联系的划分不应依赖于某个应用及职能部门的界限，可以跨越多个应用域或职能部门，否则就变成了应用策略。此外，对于信息结构不太复杂的较小单位，也可以考虑选择全局策略。

19.4.2　概念结构设计的方法和步骤

概念结构设计常用的方法是实体分析法（Entity Analysis）和属性综合法（Attribute Synthesis）。

实体分析法又称为自顶向下（top-down）方法。它从总体概念入手，从分析一个单位的事务活动开始，首先识别用户所关心的实体及实体间的联系，建立一个初步的数据模型框架，再用逐步求精的方法加上必需的描述属性，形成一个个完整的局部数据模型，称为用户视图，最后将这些视图集成为一个统一的数据模式，称为用户视图的集成。这种统一的数据模式（即全局信息结构）通常用 E-R 图表示。可见，实体分析法是通过 3 个不同的步骤完成概念模型设计的：建立用户视图（局部信息结构）；视图集成为统一的数据模型（全局信息结构）；用 E-R 图描述全局信息结构。这种方法的优点是：减少了分析中所涉及的对象数，简化了分析过程，且采用图形表示法，因而更为直观，易于理解，有利于用户在设计过程中的介入，所以被广泛采用。

属性综合法又称为自底向上（bottom-up）方法。其基本点是将需求分析中收集的数据元素作为分析对象，高层实体及联系通过低层属性综合而成的设计技术。实际上，这是一种基于统计分析推导的方法，即通过对数据元素与应用任务联系的定性定量统计分析技术来推导出相应的信息结构。其处理过程可分为属性分类、实体构成、联系的确定等相对独立的步骤。这种方法适用于较为简单的设计对象而不适于稍为复杂的应用环境，因为要对几百甚至数千个数据元素进行综合处理就不容易，何况在数据分析的前期阶段，数据库管理员通常并不完全清楚在数据库模式中究竟包含哪些数据元素。

19.4.3　数据抽象和局部视图设计

数据模型是数据库系统的核心和基础，各种机器上实现的 DBMS 软件都是基于某种数据模型的且有一个共同的特点，因为它们是在具体的机器上实现的，所以在许多方面给出了细致严格的限制。而现实世界中应用环境是复杂多变的，各种事务的表现形式也与机器世界中的相距甚远。在进行数据库设计时，如果将现实世界中的客观对象直接转换为机器世界中的对象，就会感到非常不方便，注意力往往被牵扯到更多的细节限制方面，而不能集中在最重要的信息的组织结构和处理模式上。因此往往是将现实世界中的客观对象首先抽象为不依赖任何具体机器的信息结构，这种信息结构不是 DBMS 支持的数据模型，而是概念级模型。然后再把概念级模型转换为具体机器上 DBMS 支持的数据模型。

由于各个部门对于数据的需求和处理方法各不相同，对同一类数据的观点也可能不一

样，它们有自己的视图，所以可以首先根据需求分析阶段产生的各个部门的数据流图和数据字典中的相关数据设计出各自的局部视图。

在实体分析法中，局部视图设计的第一步是确定其所属的范围，即它所对应的用户组，然后对每个用户组建立一个仅由实体、联系及它们的标识码组成的局部信息结构（局部数据模式）框架，最后再加入有关的描述信息，形成完整的局部视图（局部数据模式）。这样做的目的是为了集中精力处理好用户数据需求的主要方面，避免因无关紧要的描述细节而影响局部信息结构的正确性。整个过程可分为如下几个步骤：

①确定局部视图的范围。

②识别实体及其标识。

③确定实体间的联系。

④分配实体及联系的属性。

1．确定局部视图的范围

需求说明书中标明的用户视图范围可以作为确定局部视图范围的基本依据，但它通常与子模式范围相对应，有时因为过大而不利于局部信息结构的构造，故可根据情况修改；但也不宜分得过小，过小会造成局部视图的数量太大及大量的数据冗余和不一致性，给以后的视图集成带来很大的困难。

局部视图范围确定的基本原则如下：

- 各个局部视图支持的功能域之间的联系应最少。
- 实体个数适量。一个局部视图所包含的实体数量反映了该局部视图的复杂性，按照信息论中“7±2”的观点，人们在同一时刻可同时顾及的事情一般在5～9之间，以6或7最适当。因此，一个局部视图内的实体数不宜超过9个，否则就会过于复杂，不便于人们理解和管理。

2．识别实体及其标识

在需求分析中，人们已经初步地识别了各类实体、实体间的联系及描述其性质的数据元素，统称为数据对象，它们是进一步设计的基本素材。这一步的任务就是在确定的局部视图范围内，识别哪些数据对象作为局部视图的基本实体及其标识，定义有关数据对象在E-E-R 模式中的地位。

1）数据对象的分类

为了确定数据对象在局部 E-E-R 数据模式中的地位，首先必须对所有已识别的数据对象加以适当的分类，以便于根据它们所属的对象类来确定它们在相应局部 E-E-R 模式中的身份。数据对象分类的原则是同一类中的对象在概念上应该具有共性。例如，高校中的教师这个概念是指在职的教学人员，他们的性质由姓名、性别、出生年月、工作单位、职称、专业特长等数据项加以描述。因此，教授、副教授、讲师和助教均可归入教师这一类，但他们在概念上并不完全相同，除共性之外，还各有其特殊部分，如教授、副教授有研究方向、指导研究生等描述项，职称也不一样，因此存在分类层次问题，为此可运用面向对象数据模型中子类与超类的概念。

2）识别实体与属性

建立局部 E-E-R 数据模式，还须识别每个对象类在局部 E-E-R 模式中的地位：实体、

属性或联系。实体和属性之间在形式上的界限并不明显，常常是现实世界对它们已有的大体的自然区分，它随应用环境的不同而不同。在给定的应用环境中区分实体和属性的总的原则是要在此应用环境中显得合理，且同一个对象类在同一局部视图内只能作为一种成分，不能既是实体又是属性或联系。此外，为了对已给定的需求目标，更合理地确认局部 E-E-R 模式中的实体和属性，以便在逻辑设计阶段从 E-E-R 模式得到更接近于规范化的关系模式，可按如下一般规则来区分实体与属性。

①描述信息原则

在 E-E-R 模式中的实体均有描述信息，而属性则没有。据此，可将一个需要描述信息的对象类作为实体，而将只需有一个标识的对象类归为属性。例如，仓库这个对象类在某些事务处理中需要用到仓库的面积、地点、管理员姓名等描述信息，则宜将其归入实体。但如人的年龄、物品的重量等对象类，在一般应用中都不需要描述信息，所以它们在通常情况下都作为属性。

②多值性原则

所谓多值性是指若一个描述存在多个值描述其描述对象，则即使该描述项本身没有描述信息也宜划为实体。例如，加工种类与其描述的工件之间就符合多值性原则，因为每个工件实体可能需要多个工种的加工，尽管工种除工种名外不需要其他的描述，但还是将工种作为实体好。需要注意的是，这一原则最好与存在性原则结合起来使用，如果将被描述对象删除后，描述项没有再存在的意义，则即使此描述是多值的也不宜作为实体。例如，零件的颜色可以是多值的，但颜色离开了其描述的对象就没有单独存在的意义，因此还是作为属性为宜。

③存在性原则

设对象类 R 的描述 A 和 B 的值集分别为$\{a_1，a_2，\cdots，a_n\}$和$\{b_1，b_2，\cdots，b_n\}$。若从 R 中去掉 B 的某个值 b_i，从而去掉与它的所有联系，如果 b_i 的消失对应用不产生任何影响，则 B 作为属性，否则 B 应作为实体。

④多对一联系性

属性不再与其描述对象以外的其他对象类发生联系。相反，如果一个对象类的某个描述项与另一个对象类存在着多对一联系，则此描述项即使本身没有描述信息，也宜将其作为实体。例如，前面讲的工件与加工种类的例子中，如果还有一个车间实体，加工种类与车间之间存在多对一联系，因此将加工种类划为实体更合适。

⑤组合标识判别原则

若一个对象类的标识是由其他对象类的标识组成的，该对象一般应定义为联系。相反，如果组成某对象类标识的各成分不是其他实体的标识，且作为实体在应用的上下关系中很自然，则可以定义为实体。例如，一个工厂生产的零件须由名称及规格组成组合标识，在一般应用中应将零件作为实体较为适宜。

实体与属性的识别过程是一个相互作用的反复过程，随着实体的确认，属性也将趋于明朗，反之亦然。在此过程中根据应用需求对已经识别的实体和属性做适当指派，发现问题再来调整，在识别过程中必须遵循前面所讲的总原则。

3）*对象的命名*

在需求分析中得到的数据对象通常都是有名称的，但由于这些名称未经规范化，常常存在诸如同名异义、异名同义等许多命名上的冲突、不一致性及语义不清等问题，是造成数据不一致性及数据冗余的一个重要原因，此外，数据对象原有的名称有的过于冗长，给使用带来很大不便。为此，需要按一定的规范对每一类数据对象重新命名，给它指定一个简洁明了且唯一的名字，避免异义同名和异名同义存在。命名规范一般包含如下原则和规定。

- 数据对象名应清晰明了便于记忆，并尽可能采用用户熟悉的名字。
- 名字要反映数据对象的主要特点并力求简洁，以利于减少冲突和方便使用。
- 遵守缩写规则。缩写规则包括一般缩写规则和系统中专用名称的缩写规定，对于较大的复杂系统应编制缩写字典，便于参照。凡有缩写规定的不得使用全称，以免混淆。
- 规定统一的命名约定并加以遵守。例如对属性的命名可以采用如下形式的约定：

实体名·分类词——修饰词

其中，分类词指单位内通用的数据项名，如名称、号码、小计、总计、合计、单位、摘要、日期、时间等。每个单位应有一个标准的分类词表，加于分类前面的实体名和后面的修饰词可用以说明该分类词在特定的上下文中具有的特殊含义。例如，合同上的数据项日期可命名为：

合同·日期——年月

制定命名约定的基本原则是简明一致、避免混淆，具体可根据单位内数据对象的复杂程度自行制定。

在完成了实体与属性的识别后，必须按照命名规范仔细地审核每类数据对象的名字，纠正不符合规范的命名，务必使每个对象名符合规范要求。

4）*确定实体的标识*（identifier）

实体的标识是能够唯一地标识一个实体的属性或属性组，也就是该实体的关键字。确定实体标识在实体识别与规范化命名之后进行，首先确定每个实体的候选关键字。一个实体可能有若干候选关键字，选择其中对有关用户最熟悉的一个候选关键字作为主关键字（或主码），并将每个实体的候选关键字、主关键字记入数据字典。

3．确定实体间的联系

实际上，识别联系的主要任务是在需求分析阶段完成的。这里的工作一是从局部视图的角度进行一次审核，检查有无遗漏之处；二是确切地定义每一种联系。

现实世界中的诸多形式的联系大致可分为 3 类：存在性联系、功能性联系和事件联系。

存在性联系如学校有教师，教师有学生，工厂有产品，产品有顾客等；功能性联系如教师讲授课程，教师参与科研，仓库管理员管理仓库等；事件联系如学生借书、产品发运等。

根据上述分类仔细检查在给定的局部视图范围内是否有未识别的联系，在确认所有的联系都已识别并无遗漏之后，还须对联系进行正确的定义。定义联系就是对联系语义的仔细分析，识别联系的类型，确定实体在联系中的参与度。

1）二元联系的类型与定义

二元联系是指两个实体类之间的联系。根据参与联系的两个实体类值之间的对应关系分为一对一、一对多及多对多 3 种类型。对每一种联系类型，要确定实体在联系中的参与度，并以 m:n 的形式标在 E-E-R 图上要说明的实体旁。若 $m>0$，表明该实体参与联系是强制性的，若 $m=0$ 则是非强制性的。下面分别讨论上述 3 类联系的定义。

①一对一联系

这是一种最简单的联系类型。若对于实体集 A 中的每一个实体，实体集 B 中至多有一个实体与之联系，反之亦然，则称实体集 A 与实体集 B 具有一对一联系，记为 1:1。例如，在一个施工单位中，如果规定每项工程最多只能由一名工程师负责管理，而一名工程师最多也只能负责一项工程，则工程师与工程间的这种管理联系便是一对一联系。按照实体参与联系的强制性情况，又可分为如下 3 种情况。

（a）两类实体都是强制性的

假如规定每个工程师一定要负责一项工程，每项工程也一定要有一位工程师负责，便属于此种情况。如果工程师的标识为职工号，工程的标识为工程号，对于工程师与工程间的 1:1 联系，可用职工号或工程号作为标识。

（b）其中仅有一类实体是强制性的

若规定每项工程必须由一名工程师负责，但并不是所有工程师都必须负责一项工程（因为有可能没有那么多工程），此时，每一项工程一定对应着唯一负责联系，所以工程号可用作联系的标识。

（c）两类实体均为非强制性的

如果工程师不一定安排负责管理工程，有的工程项目暂时可以不安排工程师负责管理，这种情况表示凡分到工程项目的工程师与工程项目之间总存在一一对应的联系，因此职工号或工程号均可选为负责联系的标识，定了其中一个为标识，另一个就作为候选关键字。

②一对多联系

若对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体（$n\geqslant0$）与之联系；反之，对于实体集 B 中的每一个实体，实体集 A 中至多只有一个实体与之联系，则称实体集 A 与实体集 B 有一对多的联系，记为 1:n。以专业与学生间的关系为例，如规定一个专业可以管理许多学生，每个学生只能属于一个专业，这种联系就是一对多联系。对这种联系只需关心“多”端实体的强制性，分两种情况进行讨论。

（a）“多”端的实体是强制性的

此时，每个学生必须归属某个专业，即每个学生都有一个确定的专业，但每个专业都不唯一地对应一个学生，故可以选择学号作为联系的标识。

（b）“多”端的实体是非强制性的

对本例而言系指有些学生（如新生）不属于任何专业的情况。此时实际上仅表示已经分配专业的学生与专业之间的联系，对这些学生中的每一个都有一个确定的专业，因此，应以学号为联系的标识，而专业代号只作为联系的一般属性。

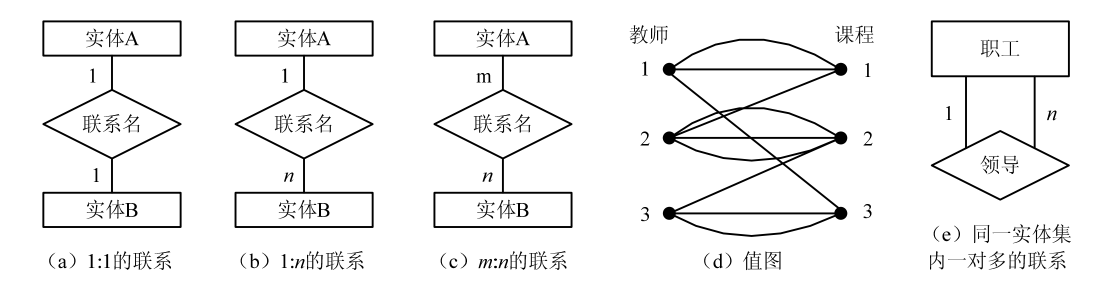
③多对多联系

若对于实体集 ***A*** 中的每一个实体，实体集 ***B*** 中有 n 个实体（$n \geqslant 0$）与之联系，反过来，对实体集 ***B*** 中每一个实体，实体集 ***A*** 中也有 m 个实体（$m \geqslant 0$）与之联系，则称实体集 ***A*** 与实体集 ***B*** 具有多对多联系，记为 m:n。教师与学生这两个实体类间的教与学的联系就是多对多的联系。这时，只有<教师，学生>对才能确定一个特定的教学联系。因此，一般情况下可以用两个关联实体的标识拼凑（concatenate）作为联系的标识，但这种方法对某些情况就不能构成有效的联系标识。当一个实体值在同一个联系上可能存在多个不同的联系值时，就会出现这种情况。如教师与其讲授的课程之间的联系，同一个教师可讲授几门不同的课程，也可以多次讲授同一门课程，这是一种特殊的多对多联系，这种情况可用如图 19-3（d）所示的值图（Occurrence Diagram）表示。值图是表示具体的实体及其关联的一种图示法，其中圆点表示具体的实体，连线表示实体间的联系。而通常的 E-E-R 图或 E-R 图只能表示型而不表示值，所以称为型图（Type Diagram）。显然，对于与讲授课程间的联系，如在教师档案中要求记录担任教学工作的情况，就需要在联系标识中增加表示授课日期的属性，即其合适的联系标识可能为（教师号、课程号、授课日期）。

④实体类内部的联系

这种联系发生在同一类实体的不同实体之间，因此称为内部联系或自联系，它也是一种二元联系，其表示方式与前面的二元联系并无不同，要注意的是，仔细区别同一实体类中的不同实体在联系中扮演的不同角色及联系标识的选择。例如，在职工类实体中间就存在着管理者与被管理者的联系。一个职工可以管理别的职工，称为管理者或领导者。一个管理者可以管理多个职工，而一个职工最多只从属于一位管理者，从而构成了一对多联系。若规定所有职工都要受管理（最高管理者考虑自己管理自己），但不是所有职工都是管理者，则此联系在“多”端呈现强制性。其中每个联系实体包含两个职工号值：职工号和管理员职工号，以区别不同实体在联系中的角色。

若略去实体与其属性图，上述实体间的联系可用图 19-3 表示。



（a）1:1的联系　（b）1:n的联系　（c）m:n的联系　（d）值图　（e）同一实体集内一对多的联系

图 19-3　实体间的联系

2）多元联系的识别与定义

两个以上的实体类之间的联系称为多元联系。例如，在供应商向工程供应零件这类事件中，如果任一供应商可向任一工程供应任一种零件，则为了确定哪个供应商向哪个工程供应了何种零件，就必须定义一个三元联系，因为只有供应商、工程及零件三者一起才能唯一地确定一个联系值。其联系的标识由参与联系的实体类的标识拼接而成，在此例中由供应商、工程、零件 3 个实体类的标识拼接而成。

需要注意的是，涉及多个实体的事件是否属于多元联系完全取决于问题的语义，不可一概而论。例如，如果上例中的问题说明变成每个工程需要订购一定的零件，而任一供应商可向任一工程供应零件。这里有两层意思，一是只有工程确定了才能确定订购的零件，二是只有供应商及工程确定了，才能确定一个供应关系。根据这一情况，应定义两个二元联系，如图 19-4（a）所示。

假如问题的说明是任一供应商向任一工程供应零件，但某个供应商向某项工程供应的零件是一定的，则在供应商与工程之间的关系确定后，供应的零件也就确定了。由此可知，只需定义一个二元联系就可以，如图 19-4（b）所示，其中供应的零件作为供应联系的一个属性。

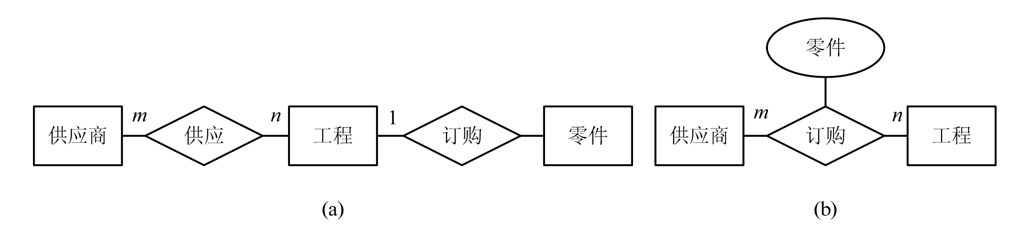


图 19-4　多元联系

总之，具体问题应该具体分析，以便使定义的模式确切地表达问题的语义。

3）消除冗余联系

若出现两个或两个以上的联系表示的是同一概念，则存在着冗余的联系，具有冗余联系的 E-E-R 模型转换为关系模型可能会得到非规范化的关系，因此必须予以消除。

出现冗余联系的一个重要原因是存在传递联系。例如，图 19-5 中表示了产品与零件之间的“组成”联系，零件与材料之间的“消耗”联系及产品与材料间的“使用”联系。其中，材料与零件间的联系是 $1:n$，零件与产品之间的组成联系是 $n:m$，其实由这两个联系必然得出产品与材料之间的使用联系 $m:n$。因此，图中产品与材料间的联系是冗余的，应将其去掉。

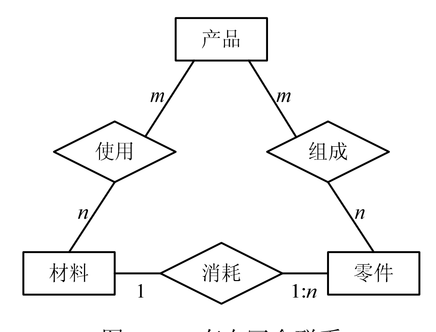


图 19-5　存在冗余联系

4）警惕连接陷阱（Connection Traps）

连接陷阱是一种存在语义缺陷的联系结构，它是由于在定义联系过程中对语义的理解出现偏差而造成的，因而无法由它得到需要的信息。连接陷阱可分为扇形陷阱、断层陷阱和深层的扇形陷阱 3 种情况。

①扇形陷阱（Fan Traps）

两个实体类间的一对多联系，由一个实体值引出多个同一类型的联系值，其值图是一种扇形结构，故称为扇形联系。扇形陷阱则指由一个实体引出两种不同类型的扇形联系，形成双扇形结构。图 19-6（a）是这种结构的一个例子，图 19-6（b）是一个值图。从图 19-6（b）值图可以看出，从这种联系结构无法获得哪个职工属于哪个专科的信息，其原因是将专科与职工之间的联系通过医院来连接的，如采用图 19-7（a）所示的联系结构，图 19-7（b）是它的值图。新的结构能较自然地表示医院、专科及职工之间的层次关联。如果假定任一医院的职工无例外地分属于医院的各个专科，该结构可以确定一个职工所属的专科或医院，但如果允许某些职工直属医院而不属于任何专科，那么这种结构还是不合适的。

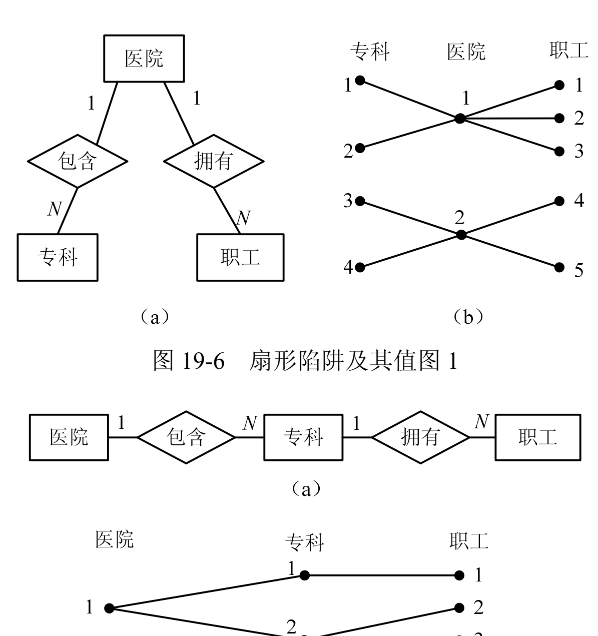


图 19-7　扇形陷阱及其值图 2

②断层陷阱

断层陷阱是指因为型图所含的传递联系而掩盖了某些特定的直接联系的现象。例如，图 19-7（a）的联系结构虽然隐含了医院与其职工的联系（传递联系），但却没有提供部分职工直属于医院的联系路径，因而出现了断层陷阱。解决办法是设置一个虚构的专科或增加一个联系，如在本例中增加一个医院—职工联系。增加虚构实体和现实世界不符，因此可考虑增加一个联系，但增加新的联系在某些情况下也可能带来新的麻烦，见下面的讨论。

③深层的扇形陷阱

以一个“教师指导学生参加课题”的例子来说明，若每个学生可在多位教师指导下参加多个课题研究，每位教师可指导多名学生，但现在只允许一名教师指导一名学生参加一个课

题，不允许多名教师指导同一名学生参加某个课题。对此可先建立一个由两个多对多的二元联系组成的模式，如图 19-8（a）所示。利用联系的分解法则将其分解为如图 19-8（b）所示的结构，它是以学生为中心的双扇形结构，图 19-8（c）是它的值图，从该联系结构无法得到哪位教师指导哪个学生参加何课题的信息，这表明存在扇形陷阱。图 19-9（a）是对它的一种改进，在其中增加了一个教师与课题的联系。该联系结构能确切地提供“教师 1 指导学生 1 参与课题 1”，“教师 2 指导学生 2 参与课题 1”的信息，但从此图无法确定教师 1 指导学生 2 参与了哪一个课题。对教师 2 和学生 1 也是如此，因为参加课题 1 或 2 均是正确的语义。之所以如此，原因在于新增加的教师与课题间的多对多联系带来了两个新的双扇形结构，即以教师为中心的及以课题为中心的双扇形结构。增加的新联系虽然消除了原来的陷阱，却产生了新的陷阱。对这类问题的有效解决办法是将 3 个实体间的联系定义成一个单一的三元联系，它的 E-E-R 模式如图 19-9（b）所示。现在，每一个联系值唯一地确定了另一个联系值，从中可获得哪位教师指导某学生参加哪一课题的确定信息。可见问题的实质是对于应该定义成三元联系的问题千万不能用二元联系代替。

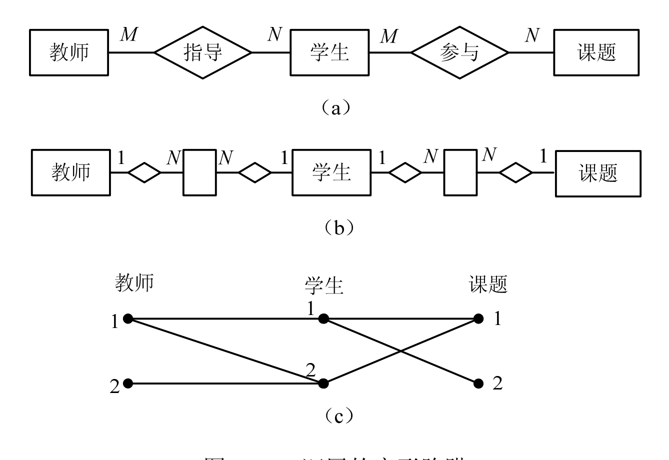


图 19-8　深层的扇形陷阱

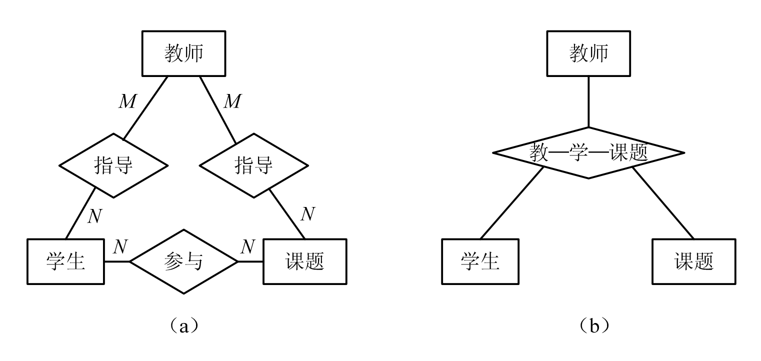


图 19-9　改进的扇形结构

4.分配实体及联系的属性

在需求分析中收集的数据对象集内，除去已识别的实体、联系及标识外，剩下的主要是非标识属性。问题的实质就是将这些非标识属性恰当地分配给有关实体或联系。不过分配这些属性时，应避免使用用户不易理解的属性间的函数依赖关系及其有关准则，而应该从用户需求的概念上去识别框架中实体或联系必须有的描述属性，并按下述两条原则分配属性。

1）非空值原则

所谓非空值原则是指当一个属性的分配在几个实体或联系中可以选择时，应避免使属性值出现空值的分配方案。例如，前面曾经讲过工程师负责工程的模式中，其联系是一对一且两个实体类均属非强制性的情况。现有一个属性项“施工期限”，按依赖关系考虑可加给工程师、工程或负责联系三者中的任何一个，因为工程师与工程在这里是一对一的联系，工程师定了，工程的施工期限也定了。但有的工程师可能没有分配到负责工程的任务，若把施工期限作为工程师的属性，在此情况下便会出现空值；若作为工程的属性，则工程可能还未纳入计划，也会出现空值，可见将“施工期限”分配给负责联系最为适宜。

2）增加一个新的实体或联系

在分配属性过程中，有时会出现有的数据项在框架模式中似乎找不到适合依附的实体或联系的情况。对于这种情况，常常可通过在原模式中增加一个新的实体或联系加以解决。例如，图 19-10 所示为病区/病人模式，这是一个一对多，且“多”端为强制性的联系，所有明显的属性均已分配完后，尚有属性项手术名及手术号待分配。这里手术名依赖于手术号，一个病人可能接受多种手术，一个病区可能接纳不同种手术的病人。在此情况下，手术号与手术名两个数据项不论作为哪个实体或联系的属性均不合适，为此可以在原来的模式中增加一个新的实体——手术，再加一个病人与手术的联系即可，如图 19-11 所示。

按上述属性分配原则建立的模式将有利于转换成规范化的关系模式。

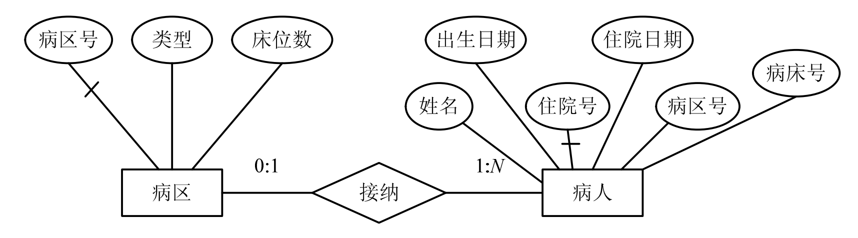


图 19-10　病区/病人模式

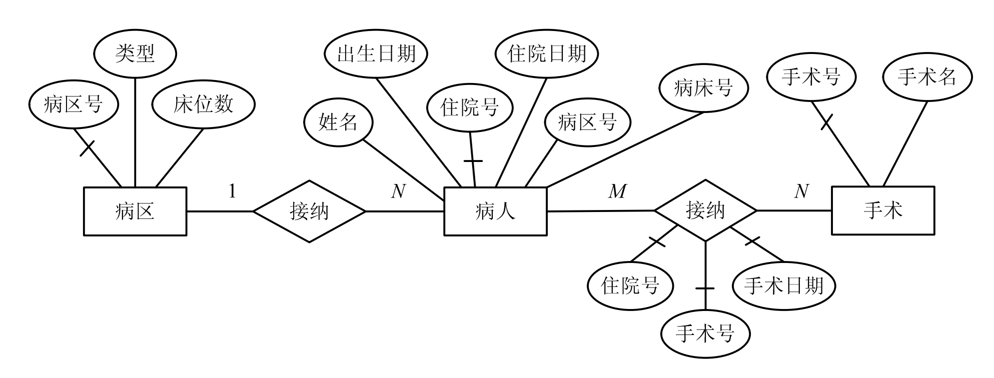


图 19-11　改进后的模式

在属性综合法中，其高层的实体、实体间的联系是通过对底层数据元素的分析、归类和聚集而逐步形成的。从具体的设计过程而言，属性综合法是一种统计分析的方法，它利用统计数据元素与事务处理的关系导出结果。此法的优点是只要有一本好的数据字典和一张能反映用户需求的数据流程图，设计者即使不太熟悉业务，也能应用这种方法开展工作。其缺点是当应用环境比较复杂、数据元素较多时，很难用人工进行统计分析。

5.属性综合法构造局部 E-R-R 数据模式的步骤

这里简要介绍在局部视图范围确定后，属性综合法构造局部 E-E-R 数据模式的基本步骤，它共有 5 步，通常称为“五步”法。

1）数据元素的归类

数据元素归类的目标是识别实体和联系，并把属性归类成实体的属性和联系的属性。

①数据元素归类的基本原则

具有如下特征的数据元素一般属于实体。

- 被大量的事务处理所使用。
- 与大量的其他数据元素一起使用。
- 与其他单个数据元素一起使用的次数少于被所有事务处理交叉使用的总次数，亦即前者的使用率较低。

由两个以上的数据元素组成的集合，若满足如下条件，则可作为联系的候选者。

- 集合中的每个数据元素都可归类为实体。
- 集合中每个数据元素的使用频率均较高。
- 当数据元素集合在交叉事务处理中出现时，与其他数据元素一起使用的次数较多。

具有如下特征的数据元素，一般是一个与实体相关的属性。

- 使用它的事务处理相对而言较少。
- 它仅同少量的其他数据元素一起使用。
- 一般而言，与其他元素一起使用的次数与它被所有事务处理交叉使用的总次数相比，使用率较高。

具有如下特征的数据元素，一般为与联系相关的属性。

- 有半数的事务处理要使用它。
- 与半数的其他数据元素一起使用。
- 它同其他数据一起使用的次数与它被所有事务交叉使用的总次数相比，使用率是平均的。

②数据元素的归类的具体步骤

（a）确定单位中事务处理/数据关系

其基本方法是将局部视图范围内部的事务处理与涉及的全部数据元素分别编号，在此基础上分别列出每个事务处理使用的数据元素号。

在建立事务处理/数据关系时，不要把事务处理不使用的数据元素也包括进去。对于异名同义的数据元素，应用一个具有代表性的数据元素代表之，且一个数据元素若在一个事务处理中被多次使用，在事务处理/数据关系中只写一次。这步工作最好重复一二次，以便更接近于实际情况，保证统计分析结果的正确性。

（b）定义事务处理使用向量、关系矩阵、全体数据关系向量

- 定义事务处理使用向量

事务处理向量的初值为零，以后根据各个事务处理对数据元素的使用逐一修改事务处理使用向量。办法是：对每个事务处理使用的数据元素都要在事务处理向量的相应单元中

加 1，对每个事务处理重复上述修改操作，最后得到的事务处理使用向量表示了使用每个数据元素的事务处理个数。

- 定义关系矩阵

关系矩阵用于反映各个事务处理数据元素间的所有数据关系，它的行和列均为数据元素的编号（即引用号），其初始值为 0。关系矩阵的填法是：取出某一事务处理中的任一个数据元素的引用号作为行号，再分别把该事务处理中其他数据元素的引用号作为列号，将由此确定的关系矩阵单元的值加 1。例如，设有事务处理 1 的数据元素是 1、3、7、14、20、21，在关系矩阵中的填入过程是，将数据元素 1 的引用号 1 作为关系矩阵的行号，再把事务处理的其他数据元素引用号 3、7、14、20、21 作为列号，将关系矩阵中相应单元的值加 1。然后，再填入数据元素 3 的关系，方法同前，依次类推，直到事务处理 1 中的所有数据元素的关系填完为止。在事务处理 1 的数据元素填完后，用同样的方法依次填入其他事务处理的数据元素的数据关系。

- 定义全体数据关系向量

全体数据关系向量的形式与事务处理使用向量相类似，不过其中每个单元的值是关系矩阵中相应行中非零单元的个数。

（c）生成使用率向量

利用关系矩阵与事务处理使用向量可以构造一个包含每个数据元素使用率的向量。

（d）确定数据归类标准

从上面得到的事务处理使用向量、全体数据关系向量和使用率向量 3 个向量的基础上画出相应的直方图，使用分界点取法，确定数据元素下面 3 类特征的高值、平均值和低值：

- 交叉使用事务处理。
- 与其他数据元素一起使用。
- 使用率。

根据事务处理/数据关系的个数和数据的范围，划分成 3 个不同的分界点。划分的结果若与实际情况存在很大的差别就应回过头去重新选择参数和重新计算数据归类标准。

根据数据归类标准与各个数据和元素在各种不同情况下的出现值，采用上述统计分析结合数据元素的归类基本原则，可初步地分出一些实体、联系和属性。但由于统计分析方法常常受数据选取范围、数量的大小等因素的影响，往往不够准确，仅给出一些定量的信息作为参考。如欲精确地进行数据归类，必须与语义相结合，根据实际情况而定。

2）*形成实体及其标识*

在上述分析得到的数据元素初步归类的基础上，再结合语义，就可标明实体及其标识。

3）*分析实体间的联系*

联系是由两个或两个以上的实体来标识的，因此通过分析实体属性间的联系，并结合实体分析法中定义联系的原则和方法，定义实体间的联系。若实体的标识是组合属性，则选取其中有代表性的一个。一般选取事务处理使用值较大、与其他数据一起使用值较大、使用率较低的数据元素。

至此，就可建立局部视图的 E-E-R 模式框架，并注意消去冲突和冗余联系，其方法与实体分析法相同。

4）属性的指派

结合前面得到的与数据元素相关的 3 个向量：事务处理使用向量、全体数据关系向量和使用率向量，以及数据元素归类的基本原则和数据归类的标准，建立 4 张表，归为实体属性的数据元素表；归为联系属性的数据元素表；既可归为实体属性，又可归为联系属性的数据元素表；每个数据元素与其他数据元素一起使用的百分比表。前面 3 张表的结构是一样的，均由数据元素引用号和数据元素两部分组成，仅表名和内容不同而已。

属性指派的任务就是要把归为属性的数据元素分配给每一个具体的实体类和联系。

①实体属性的指派

首先，从归为实体属性的数据元素表中取出第一个数据元素，查阅关系矩阵，看它与哪个实体一起使用过，再按如下步骤决定该指派给哪个实体。

若只有一个实体的标识属性与它一起使用，则将其归为这一标识属性所指的实体。

若与多个实体的属性一起使用，且删去低百分比值的标识属性后，至少剩下一个标识属性，则进行删去工作，再进行下一步，否则直接进入下一步。

逐一删去每一个标识属性，若删去后使用该数据元素失去存在的意义，则该数据元素就归为该标识属性所属的实体，否则就不用于该标识属性所指的实体。

然后对实体属性的数据元素表中的其他数据元素重复上述过程，就可把表中的所有数据元素指派给相应的实体。

②联系属性的指派

首先，从联系属性的数据元素表中取出第一个数据元素，从关系矩阵找出它与实体标识属性一起使用的百分比，再执行如下步骤。

（1）分析联系标识属性的组合（它涉及两个以上有关实体）。

（2）查阅联系的标识组中数据元素与其他数据元素一起出现的百分比表中出现的次数。

（3）若出现次数为一，则该数据元素应属于这次出现的数据元素的组合所代表的联系。

（4）若删去百分比低的数据元素的属性组合后，至少剩下一个标识属性组合，则进行删除后再做下一步，否则直接做下一步。

（5）分析每个标识属性组合与从联系属性的数据元素表中取出的数据元素之间的关系，若删去某组合会使该数据元素失去存在意义，则应把该数据元素归为该标识属性组合所指的联系，否则不属于该组合所指联系的属性。

对联系属性的数据元素表中的其余数据元素重复执行上述过程就完成了联系属性的数据元素表中数据元素的分配工作。

③两可属性的处理

对既可归为实体属性，又可归为联系属性的数据元素，可按“实体属性的指派”中步骤作为实体的属性处理，若找不到恰当的实体，再按联系的属性处理（方法同前），直到找到比较合适的归类。

④实体、联系及其属性的最后形成

综合前面 3 步，并根据实际业务情况对数据元素做适当调整，以形成最后结果。

5．描述信息结构

用 E-E-R 模型或选定的其他数据模型来描述前面分析得到的实体、联系及它们的属性，并确定各实体之间的类型，即形成局部视图的数据模式。

由上述讨论可知，属性综合法是以概率划界的，而划界的标准是主观确定的。因此，如果只使用一个参数，因缺乏可比性，可能会造成主观随意性而产生错误。最好是提供若干标准，以便作为参数选择，但这样做工作量很大，需要用辅助工具实现。

19.4.4 局部 E-R 模型的集成

当所有局部 E-R 设计完毕，就可开始集成工作。整体 E-R 图包含了每个局部 E-R 图的信息，它合理地表示了一个简单而又完整的数据库概念模型。

各局部 E-R 图之间的冲突主要有三类，分别是属性冲突、命名冲突和结构冲突。

（1）属性冲突。包括属性域冲突和属性取值冲突。属性冲突理论上好解决，只要换成相同的属性就可以，但实际上需要各部门协商，解决起来并不简单。

（2）命名冲突。包括同名异义和异名同义。处理命名冲突通常也像处理属性冲突一样，通过讨论和协商等行政手段加以解。

（3）结构冲突。包括同一对象在不同应用中具有不同的抽象，以及同一实体在不同局部 E-R 图中所包含的属性个数和属性排列次序不完全相同。对于前者的解决办法是将属性变换为实体或实体变换为属性，使同一对象具有相同的抽象。对于后者的解决办法是使该实体的属性取各局部 E-R 图中属性的并集，再适当调整属性的次序。

19.5 逻辑结构设计

数据库逻辑结构设计的任务就是把概念结构设计阶段设计好的基本 E-R 图（或 E-E-R 图）转换为具体机器上的 DBMS 产品所支持的数据模型相符合的逻辑结构。这一阶段是数据库结构设计的重要阶段。

数据库逻辑设计的基础是概念设计的结果，而其成果应包括为某 DBMS 支持的外模式、概念模式及其说明，以及建立外模式和概念模式的 DDL 程序。因此，进行逻辑设计前，必须了解数据库设计的需求说明和概念设计的成果（包括 E-E-R 图和其他文档），并仔细阅读有关 DBMS 的文件。数据库的外模式和概念模式是用户所看到的数据库，是应用程序访问数据库的接口。因此在数据库逻辑设计阶段，还必须提供应用程序编制的有关说明，最好试编一些典型的访问数据库的应用程序，以检验所设计的概念模式是否满足使用要求。概念模式是数据库的基础，它的设计质量对数据库的使用和性能及数据库在今后发展过程中的稳定性均有直接的影响。为了设计出能够正确反映一个项目的数据间的内在联系的好的概念模式，设计时必须正确处理各种应用程序之间、数据库性能与数据模式的合理性和稳定性之间的各种矛盾，对设计中出现的各种矛盾要求要权衡利弊，分清主次，按统筹兼顾的原则加以正确处理。

逻辑结构设计一般分为如下几个步骤：

①将概念结构向一般关系模型转化。

②将第一步得到的结构向特定的 DBMS 支持下的数据模型转换。

③依据应用的需求和具体的 DBMS 的特征进行调整与完善。

下面以常用的 E-R 模型和扩充 E-R 模型为主，针对关系数据库的逻辑设计介绍基本原则和方法。

19.5.1 E-R 图向关系模型的转换

E-R 模型主要包含实体和联系两个抽象概念，实体和联系本身还可能附有若干属性。其转换的基本原则是，实体和联系分别转换成关系，属性则转换成相应关系的属性。因此，E-R 模型向关系模型的转换比较直观，但不同元数的联系具体转换方法稍有不同，下面根据不同的情况分别讨论。

1）一对一联系（1:1）

设有两个实体 E_1 和 E_2 之间为一对一联系。此情况存在 3 种可能的转换方案。

方案 1：将实体 E_1、E_2 和联系名 R 分别转换为关系 E_1、E_2 和 R，它们的属性分别转换为相应关系的属性，即得到

E_1（$\underline{k_1}$，a）

E_2（$\underline{k_2}$，b）

R（$\underline{k_1}$，k_2，r）（k_2 为候选关键字）

其中属性下面带一横线者为关系的关键字。

方案 2：将实体 E_1 转换为关系 E_1，将实体 E_2 与联系名 R 一起转换成关系 E_2'，E_2' 的属性由 E_2 和 R 的属性加上 E_1 的关键字组成，其关键字为 k_2，k_1 为其候选关键字。转换后的关系为：

E_1（k_1，a）

E_2'（k_2，b，k_1，r），（k_1 是候选关键字）

方案 3：与方案 2 类似，不过是把实体 E_1 与联系 R 一起转换成关系 E_1'，其结果为：

E_1'（k_1，a，k_2，r），（k_2 是候选关键字）

E_2（k_2，b）

上述 3 个方案实际上可归结为转换成 3 个关系和转换成两个关系两种。如果每个实体的属性数较少，而联系的属性与两个实体之一关系又较密切．则可采用方案 2 或方案 3，其优点是可减少关系数，有利于减少联接运算提高查询效率，但如果每个实体的属性较多，且合并后，会造成较大数据冗余和操作异常，则以采用方案 1 为宜。

2）一对多联系（1:n）

这种情况存在两种转换方案，其一是把两个实体类和一个联系类分别转换成对应的关系，实体的属性转换为对应关系的属性，其标识属性即为对应关系的关键字，而联系转换得到的关系其属性由两个实体的标识属性和联系类本身的属性组成，并以多端实体类的标识属性为其关键字。其转换结果为 3 个关系。第二个方案是转换成两个关系，设少端和多端的两个实体类分别为 E_1、E_2，联系名 R。转换时，将 E_1 转换为一个关系 E_1，E_2 和 R 合起来转换成一个关系 E_2'，E_2' 的属性由 E_2 和 R 的属性加上 E_1 的标识属性组成，并以 E_2 的标识属性为其关键字。

3）多对多联系（$m:n$）

两个实体类之间多对多联系组成的 E-R 模型向关系模型转换时，将两个实体和一个联系分别转换成关系，实体类的属性分别转换成对应关系的属性，其标识属性为其关键字，由联系转换得到的关系的属性由两个实体类的标识属性和联系本身的属性组成，其关键字是由两个联系的实体类的标识属性组成的。

4）多元联系

实体类分别转换为相应的关系，3 个实体类间的多元联系转换为以该联系名为关系名的关系，关系的属性由各实体的标识属性及其联系的属性组成，并以各实体的标识属性为其关键字。

5）自联系

自联系是同一实体集的实体间的联系。例如，对于职工实体类内部有领导与被领导的联系，在部件这个实体集的实体之间有组成成分与组成者之间的联系等，均属于实体类的自联系。在这种联系中，参与联系的实体虽然来自同一实体类，但所起的作用不一样。

19.5.2 设计用户子模式

在 ANSI/SPARC 建议的数据库三级结构中，用户子模式（也称为外模式）是用户所看到的数据库的数据逻辑结构。各个用户（或用户组）可以有各自的外模式。外模式是概念模式的子集，但在结构和形式上可以不同于概念模式，甚至可采用不同的数据模型，不过一般都是同一数据模型。

关系数据库的外模式由与用户有关的基表及按需要定义的视图构成。设计外模式时，可参照概念设计中的局部 E-R（或 E-E-R）图。在关系模型中，设计外模式比较简单。

19.5.3 数据模型优化

由 E-R（或 E-E-R）图表示的概念模型转换得到的关系模式经过规范化以后，基本上可以反映一个企业数据的内在联系，但不一定能满足应用的全部需要和系统要求，因此，还必须根据需求分析对模式做进一步的改善和调整，其内容主要是改善数据库的性能和节省存储空间两个方面。

1．改善数据库性能的考虑

查询速度是关系数据库应用中影响性能的关键问题，必须在数据库的逻辑设计和物理设计中认真加以考虑，特别是那些对响应时间要求较苛刻的应用，应予以特别注意。

就数据库的逻辑设计而论，可从下列几个方面提高查询的速度。

1）减少联接运算

联接（joins）运算对关系数据库的查询速度有着重要的影响，联接的关系越多，参与联接的关系越大，开销也愈大，因而查询速度也愈慢。对于一些常用的、性能要求较高的数据库查询，最好是一元查询，这与规范化的要求相矛盾。有时为了保证性能，往往不得不牺牲规范化要求，把规范化的关系再合并起来，称为逆规范化（denormalization）。当然，这样做会引起更新异常。总之，逆规范化有得有失，设计者可根据实际情况进行权衡。

2）减小关系大小及数据量

被查询的关系的大小对查询速度影响较大。为了提高查询速度，可以采用水平分割或垂直分割等方法把一个关系分成几个关系，使每个关系的数据量减少。例如，对于大学中

有关学生的数据，既可以把全校学生的数据集中在一个关系中，也可以用水平分割的方法，分系建立关系，从而减少了每个关系的元组数。前者对全校范围内的查询较方便，后者则可以显著提高对指定系的查询速度。也可采用垂直分割的方法，把常用数据与非常用数据分开，以提高常用数据的查询速度。例如，高校中教职工档案的属性很多，有些需经常查询，有些则很少查询，如果放在一起，则关系的数据量就很大，影响查询速度，把常用属性和非常用属性分开，就可提高对常用属性的查询速度。

3）尽量使用快照（snapshot）

快照是某个用户所关心的那部分数据，与视图一样是一种导出关系（Derived Relation），但它与视图有两点不同。一是视图是虚关系，数据库中并不存储作为视图的导出关系，仅仅保留它的定义。快照则是一个由系统事先生成后保留在数据库中的实关系。二是视图随数据当前值的变化而变化，快照则不随原来关系中数据的改变而及时改变，它只反映数据库中某一时刻的状态，不反映数据库的当前状态，犹如照片只反映某一时刻的情景，不能反映情景变化一样，之所以称它为快照，原因就在于此。但它与照片又有不同，快照不是一成不变的，它可以由系统周期性地刷新（refresh），或由用户用命令刷新。刷新时用当前值更新旧值。在实际应用中，快照可满足相当一部分应用的需要，甚至有些应用就是需要快照，而不是当前值。例如，注明列出“某年某月某日截止”的统计或报表就是快照。由于快照是事先生成并存储在数据库中的，因而可大大缩短响应时间。目前不少 DBMS，如 Oracle、MS SQL Server 等支持快照。对不支持快照的 DBMS，用户也可以把需要作为实关系使用的导出关系作为一个独立关系存于数据库中，但这种做法只能供查询使用，对它们的刷新及管理由用户负责。

2．节省存储空间的一些考虑

尽管随着硬件技术的发展，提供用户使用的存储空间越来越大，但毕竟仍是有限度的，而数据库，尤其是复杂应用的大型数据库，需要占用较大的外存空间。因此，节省存储空间仍是数据库设计中应该考虑的问题，不但要在数据库的物理设计中考虑，而且还应在逻辑设计中加以考虑。数据库逻辑设计中可采取如下措施。

1）缩小每个属性占用的空间

减少每个属性占用的空间，是节省存储空间的一个有效的措施。通常可以有两种方法。

- 用编码表示属性。例如，用编码代替校名、系名、专业名称等比直接用文字表示要短得多，因而有效地节省存储空间。
- 用省略符表示。例如，用DS表示数据结构，用OS表示操作系统等。凡是能用省略符表示的尽量用省略符表示，同样能节省空间。

这两种方法的缺点是失去了属性值含义的直观性。

2）采用假属性（Dummy Attribute）

采用假属性可以减少重复数据占用的存储空间。设某关系模式 R 的属性 A 和 B 之间存在函数依赖 $A \rightarrow B$，B 的每一个值需要占用较大的空间，但 B 的域中不同的值却比较少，A 的域具有较多的不同值，则 B 的同一值可能在多个元组中重复出现，从而需要占用较多的空间。为了节省空间，可利用属性 B 的域中不同值少的特点，对 B 的值进行分类，用 B' 表示 B 的类型，则 $A \rightarrow B$ 可分解成两个函数依赖，即

$$A \rightarrow B',\quad B' \rightarrow B$$

这样，就可用 B'代替原来元组较多的关系 R 中的属性 B，而另外建立一个较小的关系 R'来描述 B'与 B 的对应关系。这里 B'在原关系 R 中起了属性 B 的替身的作用，所以称 B′为假属性。例如，在职工关系中，职工的经济状况这一属性，通常由职工号决定。一个大型企业中，职工人数多，对于工资级别、其他经济来源等，如每一职工逐一填写，就要占用较多的空间。为了节省空间可把经济状况分为几种类型，在元组较多的职工关系中用经济状况的类型代替原来的经济状况，这里经济状况的类型就是假属性。另外建立一个较小的关系来描述每种经济状况类型的具体内容。

数据库设计与数学问题求解不同，它是一项综合性工作，受到各种各样要求和因素的制约，有些要求往往又是彼此矛盾的，因此，设计结果很难说是最佳的，常常是有得有失。设计者必须根据实际情况，综合运用上述原则和有关理论，在基本合理的总体设计的基础上，做一些仔细的调整，力求最大限度地满足用户各种各样的要求。

19.6 数据库物理设计

数据库物理设计是利用已确定的逻辑结构及 DBMS 提供的方法、技术，以较优的存储结构、数据存取路径、合理的数据存储位置及存储分配，设计出一个高效的、可实现的物理数据库结构。数据库物理设计过程如图 19-12 所示。

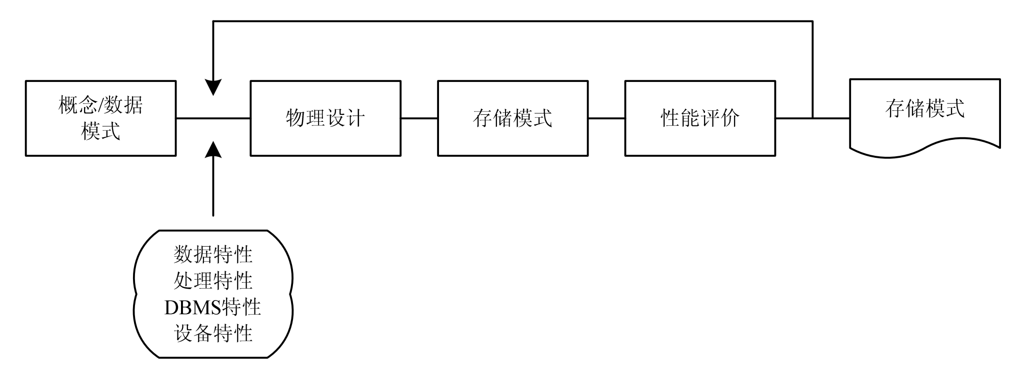


图 19-12 数据库物理设计过程

由于不同的 DBMS 提供的硬件环境和存储结构、存取方法，以及提供给数据库设计者的系统参数及变化范围有所不同，因此，为了设计出一个较好的存储模式，设计者必须了解如下几方面的问题，做到心中有数。

- 了解并熟悉应用要求，包括各个用户对应的数据视图，即数据库的外模式（子模式），分清哪些是主要的应用，了解各个应用的使用方式、数据量和处理频率等，以便对时间和空间进行平衡，并保证优先满足应用的时间要求。
- 熟悉使用的DBMS的性能，包括DBMS功能，提供的物理环境、存储结构、存取方法和可利用的工具。
- 了解存放数据的外存设备的特性，如物理存储区域的划分原则，物理块的大小等有关规定和I/O特性等。

存储模式和概念模式不一样，它不是面向用户的，一般的用户不一定、也不需要了解数据库存储模式的细节。所以数据库存储模式的设计可以不必考虑用户理解的方便，其设计目标主要是提高数据库的性能，其次是节省存储空间。

数据库物理设计内容包括记录存储结构的设计、记录集簇（Record Clustering）的设计、存取路径（Access Path）的设计和物理设计的性能评价 4 个方面。

1）记录存储结构的设计

概念模式表示的是数据库的逻辑结构，其中的记录称为逻辑记录（Logical Record），而存储模式（内模式）则是逻辑记录的存储形式，是由存储记录组成的。记录存储结构的设计就是设计存储记录的结构形式，它涉及不定长数据项的表示、数据项编码是否需要压缩和采用何种压缩、记录间互联指针的设置，以及记录是否需要分割以节省存储空间等在逻辑设计中无法考虑的问题。

2）记录集簇的设计

记录集簇的设计是通过把一些经常用于同一访问的记录在外存空间中集中存放在一起，或存放在相邻的区域，以提高数据库的性能。在有些 DBMS 中，记录的存放位置由操作系统决定，DBMS 无法控制，对这类 DBMS 在数据库的设计中也就毋庸加以考虑。但有些 DBMS，如 Oracle 和 SQL/DS 的 DBMS 提供了集簇（Cluster）功能，供数据库设计者控制记录的存放。

3）存取路径（Access Path）的设计

这是利用 DBMS 提供的文件结构、索引技术等技术手段，选择适宜的文件结构和索引技术等使得主要应用能够对每个记录型或关系进行有效的访问，即提供合适的存取路径。

4）物理结构设计的性能评价

数据库物理设计是综合性的，既要满足各个应用对性能的要求，又要适应各种物理上的限制，如时间效率、空间效率、维护代价和各种用户要求，而且数据库的性能还与计算机系统的运行环境有关。例如，计算机系统是单用户还是多用户，负荷的轻重，数据库所用的磁盘是专用还是共享等，它们均影响到数据库的性能，但这些因素及其影响在数据库设计时又很难确切估计。因此，目前，数据库物理设计主要还是采用启发式的方法，即根据一般的原则和设计要求，运用 DBMS 提供的各种手段，设计出初步方案，通过基准程序测试（Benchmark Testing）进行调整。

第 20 章

常用算法设计

根据考试大纲，算法设计是每年必考的知识点，一般以 C/C++语言的形式进行描述。本章主要讨论经常考的几种算法。

20.1 算法设计概述

算法是在有限步骤内求解某一问题所使用的一组定义明确的规则。通俗来讲，就是计算机解题的过程。在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。一个算法应该具有如下 5 个重要的特征。

- 有穷性：一个算法必须总是（对任何合法的输入值）在执行有穷步之后结束，且每一步都可在有穷时间内完成。
- 确定性：算法中每一条指令必须有确切的含义，读者理解时不会产生二义性。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。
- 输入：一个算法有0个或多个的输入，以刻画运算对象的初始情况。所谓0个输入是指算法本身定出了初始条件。这些输入取自于某个特定的对象的集合。
- 输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。
- 可行性：一个算法是可行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现。

算法设计要求正确性、可读性、健壮性、效率与低存储量。

效率指的是算法执行时间。对于解决同一问题的多个算法，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。两者都与问题的规模有关。

算法的复杂性是算法效率的度量，是算法运行所需要的计算机资源的量，是评价算法优劣的重要依据。可以从一个算法的时间复杂度与空间复杂度来评价算法的优劣。当将一个算法转换成程序并在计算机上执行时，其运行所需要的时间取决于下列因素：

- 硬件的速度。例如，使用486机还是使用586机。
- 书写程序的语言。实现语言的级别越高，其执行效率就越低。
- 编译程序所生成目标代码的质量。对于代码优化较好的编译程序其所生成的程序质量较高。

- 问题的规模。例如，求100以内的素数与求1 000以内的素数其执行时间必然是不同的。

显然，在各种因素都不能确定的情况下，很难比较出算法的执行时间。也就是说，使用执行算法的绝对时间来衡量算法的效率是不合适的。为此，可以将上述各种与计算机相关的软、硬件因素都确定下来，这样一个特定算法的运行工作量的大小就只依赖于问题的规模（通常用正整数 n 表示），或者说它是问题规模的函数。

1．时间复杂度

一个程序的时间复杂度是指程序运行从开始到结束所需要的时间。

一个算法是由控制结构和原操作构成的，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同的算法，通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该操作重复执行的次数作为算法的时间度量。一般情况下，算法中原操作重复执行的次数是规模 n 的某个函数 $T(n)$。

许多时候要精确地计算 $T(n)$是困难的，我们引入渐近时间复杂度在数量上估计一个算法的执行时间，也能够达到分析算法的目的。

定义（大 O 记号）：如果存在两个正常数 c 和 n_0，对于所有的 n，当 $n \geqslant n_0$ 时有：

$$f(n) \leqslant cg(n)$$

则有：

$$f(n) = O(g(n))$$

也就是说，随着 n 的增大，$f(n)$渐进地不大于 $g(n)$。例如，一个程序的实际执行时间为 $T(n)=2n^3+3n^2+5$，则 $T(n)=O(n^3)$。$T(n)$和 n^3 的值随 n 的增大渐近地靠拢。

使用大 O 记号表示的算法的时间复杂度，称为算法的渐近时间复杂度。

通常用 $O(1)$表示常数计算时间。常见的渐近时间复杂度有：

$$O(1)<O(\log_2 n) <O(n) <O(n\log_2 n) <O(n^2) <O(n^3) <O(2^n)$$

2．空间复杂度

一个程序的空间复杂度是指程序运行从开始到结束所需的存储量。

程序运行所需的存储空间包括如下两部分。

- 固定部分：这部分空间与所处理数据的大小和个数无关。主要包括程序代码、常量、简单变量、定长成分的结构变量所占的空间。
- 可变部分：这部分空间大小与算法在某次执行中处理的特定数据的大小和规模有关。例如，100个数据元素的排序算法与1 000个数据元素的排序算法所需的存储空间显然是不同的。

算法由数据结构来体现，所以看一个程序首先要搞懂程序实现所使用的数据结构，如解决装箱问题就使用链表这种数据结构。数据结构是算法的基础，数据结构支持算法，如果数据结构是递归的，算法也可以用递归来实现，如二叉树的遍历。经常采用的算法有迭代法、递推法、递归法、穷举法、贪婪法、分治法和回溯法等，下面对这些常用算法设计技术进行探讨，并尽量对解决各个问题所使用的算法所采用的数据结构进行比较详细的分析。

20.2 递推法

递推法实际上是需要抽象为一种递推关系，然后按递推关系求解。递推法通常表现为两种方式：一是从简单推到一般；二是将一个复杂问题逐步推到一个已知解的简单问题。这两种方式反映了两种不同的递推方向，前者往往用于计算级数，后者与“回归”配合成为一种特殊的算法——递归法。

由简单推到一般的方法，一般是从前面已知的各项（组）的值，采用层层递推，最后得到后面的某个要求的某项（组）数值。如当求解问题的规模为 N，且当 N=1 时解已知或很容易得到解。若要求规模为 n 时的解，则可以先从规模 N=1 时求解，再根据 N=1 时的解求规模 N=2 时的解，这样依次递推，可求得 $N=n-1$ 时的解，再根据规模 $N=n-1$ 时的解即可求得规模 n 时的解。递推法通常用于计算级数第 n 项的值。下面以阶乘计算为例，说明递推法的基本思想。

问题描述：编写程序，对给定的 n（n≤100），计算并输出 k 的阶乘 k!（k=1，2，…，n）的全部有效数字。

要求得阶乘 k! 的值，必定已经求得了(k-1)! 的值，依次递推，当 k=2 时，要求得的 1! =1 为已知。求得(k-1)! 的值后，对(k-1)! 连续累加 k-1 次后即可求得 k! 值。例如，已知 5! =120，计算 6!，可对原来的 120 累加 5 次 120 后得到 720。

由于 k! 可能大大超出一般整数的位数，因此程序用一个一维数组存储长整数，存储长整数数组的每个元素只存储长整数的一位数字。如有 m 位长整数 N 用数组 a[ ]存储：

$$N=a[m]\times10^{m-1}+a[m-1]\times10^{m-2}+\cdots+a[2]\times10^{1}+a[1]\times10^{0}$$

并用 a[0]存储长整数 N 的位数 m，即 a[0]=m。按上述约定，数组的每个元素存储 k 的阶乘 k! 的一位数字，并从低位到高位依次存于数组的第二个元素、第三个元素……例如，6! =720，在数组中的存储形式为：

a[0]	a[1]	a[2]	a[3]	……
3	0	2	7	……

a[0]=3 表示长整数是一个三位数，接着从低位到高位依次是 0、2、7，表示成整数 720。

该问题算法程序实现见程序 20-1。

【程序 20-1】

```
# include <stdio.h>
# include <malloc.h>
# define  MAXN  1000
void pnext(int a[ ],int k)                    /*已知 a 中的(k-1)!，求 k! */
{   int *b,m=a[0],i,j,r,carry;
    b=(int * ) malloc(sizeof(int)* (m+1));
    for ( i=1;i<=m;i++) b[i]=a[i];
    for ( j=1;j<k;j++)                        /*控制累加 k-1 次*/
    {   for ( carry=0,i=1;i<=m;i++)
        {   r=(i<=a[0]?a[i]+b[i]:a[i])+carry;
            a[i]=r%10;
```

```
                carry=r/10;
            }
            if (carry) a[++m]=carry;
        }
        free(b);
        a[0]=m;
    }
    void  write(int *a,int k)
    {   int i;
        printf("%4d! = ",k);
        for (i=a[0];i>0;i--)printf("%d",a[i]);
    printf("\n\n");
    }
    void main()
    {   int a[MAXN],n,k;
        printf("Enter the number n:  ");
        scanf("%d",&n);
        a[0]=1;
        a[1]=1;
        write(a,1);
        for (k=2;k<=n;k++)
        {   pnext(a,k);
            write(a,k);
            getchar();
        }
    }
```

20.3 递归法

递归是一种特别有用的工具，不仅在数学中广泛应用，在日常生活中也常常遇到。例如，一个画家画的如图 20-1 所示的画便是一种递归的图形。

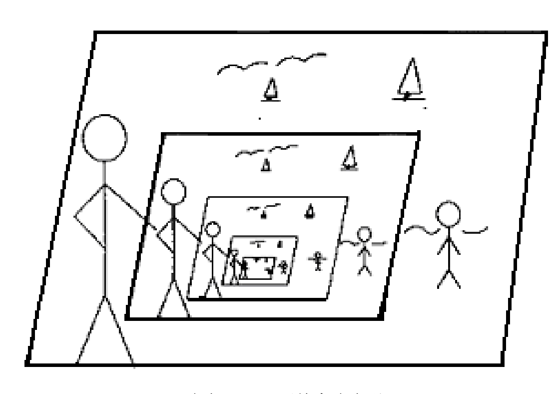


图 20-1　递归图形

递归是设计和描述算法的一种有力的工具，由于它在复杂算法的描述中被经常采用，能采用递归描述的算法通常有这样的特征：为求解规模为 N 的问题，设法将它分解成规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模较大问题的解。特别的，当规模 N=1 时，能直接得解。

递归算法包括“递推”和“回归”两部分。递推就是为得到问题的解，将它推到比原问题简单的问题的求解。如 $f(n)=n!$，为计算 $f(n)$，将它推到 $f(n-1)$，即 $f(n)=nf(n-1)$，这就是说，为计算 $f(n)$，将问题推到计算 $f(n-1)$，而计算 $f(n-1)$比计算 $f(n)$简单，因为 $f(n-1)$比 $f(n)$更接近于已知解 0！=1。

使用递推时应注意如下条件。

- 递推应有终止的时候。例如$n!$，当n=0时，0！=1为递推的终止条件。所谓“终止条件”就是在此条件下问题的解是明确的，缺少终止条件便会使算法失效。
- “简单问题”表示离递推终止条件更为接近的问题。简单问题与原问题解的算法一致，其差别主要反映在参数上。例如，$f(n-1)$与$f(n)$其参数相差1。参数变化，使问题递推到有明确解的问题。

回归是指当“简单问题”得到解后，回归到原问题的解上来。例如，当计算完 $f(n-1)$后，回归计算 $nf(n-1)$，即得 $n!$ 的值。

使用回归应注意如下问题。

- 递归到原问题的解时，算法中所涉及的处理对象应是关于当前问题的，即递归算法所涉及的参数与局部处理对象是有层次的。当解一个问题时，有它的一套参数与局部处理对象。当递推进入一个“简单问题”时，这套参数与局部对象便隐蔽起来，在解“简单问题”时，又有它自己的一套。但当回归时，原问题的一套参数与局部处理对象又会活跃起来。
- 有时回归到原问题以得到问题解，回归并不引起其他动作。

例如计算 $n!$，其公式为：

$$n! = \begin{cases} 1 & \text{当 } n=0 \\ n(n-1) & \text{当 } n\neq 0 \end{cases}$$

程序片段见程序 20-2。

【程序 20-2】

```
 int factorial(int n)
{
   if(!n) return (1);
   else  return (n*factorial(n-1));
}
```

图的深度优先搜索、二叉树的前序、中序和后序遍历等可采用递归实现。

20.3.1 斐波那契（Fibonacci）数列

问题描述：编写计算斐波那契（Fibonacci）数列，数列大小为 n。

无穷数列 1，1，2，3，5，8，13，21，35，…，称为斐波那契数列，其递归定义如下：

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

由斐波那契数列的递归定义可知，当 n 大于 1 时，这个数列的 n 项的数值是它前面两项之和。递归算法的执行过程分递推和回归两个阶段。在递推阶段，程序把较复杂的问题（规模为 n）的求解推到比原问题简单一些的问题（规模小于 n）的求解；在回归阶段，程序由在规模很小时求得的解得到较复杂问题的解。因此，对于斐波那契数列的求解，要得到数列第 n 项的值，就必须求得数列第 n-1 项的值，同理，要求得数列 n-1 的值，就必须求得数列 n-2 项的值，依次递推下去，最后需要求得数列第 1 项和第 0 项的值，递推部分结束。又当 n 等于 0 和 1 时，其数值为 1，根据这些值可求出数列第二项的值，再根据数列第二项的值可得到第三项的值，依次回归，最后可依次得到数列第 n-2、n-1、n 项的值。由这个例子，可以很清楚地了解递归的形式和过程。该问题程序实现见程序 20-3。

【程序 20-3】

```
# include< stdio.h >
int  F(int n)
{   if(n= =0)return  1;
    if(n= =1)return  1;
    if(n>1)return  F(n-1)+F(n-2);        /*递归*/
}
main(){
int i,n,m;
printf("please input n: \n");
scanf("%d",&n);
printf("the Fibonacci is : ");
  for(i=0;i<=n;i++){
    m=F(i);
    printf("%d,",m);
}
}
```

20.3.2 字典排序问题

问题描述：本程序将一段以“*”结束的文本中的单词按字典序打印出来，并且打印出该单词在正文中的出现次数。

考虑到二叉排序树可以很容易地以递归方式实现，我们使用二叉排序树来实现文本中单词按字典序排序。程序中使用二叉排序树存入单词，即每次从文件中读出一个单词，都将其按单词的字典顺序存入一个二叉排序树，第一个存入的单词为二叉排序树的根。读完文件中的单词后，中序遍历打印出二叉排序树中存放的各个单词。

为了使存储空间使用更加合理且能够处理任意长度的单词，程序设立了一个数组 text，

所有读入的单词都放在 text 数组中。函数 getword 完成读入单词的操作，并返回所读入的单词的长度。函数 insert 完成在二叉排序树中插入一个新结点的操作并返回指向二叉树根结点的指针。

该问题算法程序实现见程序 20-4。

【程序 20-4】

```
# include< stdio.h >
# include< malloc.h >
char ch=' ' ;
typedefine struct node * tree
struct node
 { char * data ;
   int  count ;
   tree  lchild ;
   tree  rchild ;
} ;
getword (word )                  /*读取单词子程序，返回单词字符个数*/
char * word ;
{ int i = 0 ;
 if (ch = = '*') return (0) ;
 while ( ch = = '' | | ch = =' \ t'  | |ch= =' \n'  )
   ch = getchar ( );              /*去掉前导空白符或其他非法操作*/
  while (ch! = '  ' && ch !=' \ t' &&ch !=' \n' && ch !=' #' )
{ word[i++]=ch;                 /*输入单词字符存入数组 word*/
     ch = getchar ( );           /*输入下一个字符*/
   }
  word [i] = 0;                 /*单词末尾用字符 0 表示结束*/
   return (i);                  /*返回单词个数*/
  }
tree insert ( root , x)               /*将数组 x 中的单词插入到二叉排序树中*/
tree root;
char * x;
 { tree p ;
int res;
if ( root = = NULL )              /*若排序树根结点为空，则置 x 为根结点*/
   { p = ( tree ) malloc ( sizeof ( * p ) );
     p->data = x;
     p->count=1;
     p->lchild = NULL ; p->rchild = NULL;
     return ( p); }                  /*根结点返回*/
else if ( ( res = strcmp ( x, root->data ) ) < 0 )     /*若 x 小于根结点数据
域单词，则搜索其左子树*/
      root->lchild=insert(root->lchild,x);       /*递归搜索左子树*/
else if(res>0)
      root->rchild=insert(root->rchild,x);       /*递归搜索其右子树*/
```

```
    else
       root->count++;                              /*若 x 等于根结点，则根结点计数器加 1*/
       return (root);
       }
    print_tree (root);                               /*打印二叉排序树*/
    tree root;
    {
    if(root!=NULL)
    { print_tree(root->lchild);
       printf("%d %s \n", root->count, root->data);
       print_tree(root->rchild);
    }
      }
       main()
    {  int len;
       char *word,*text;
       tree  root;
       root=NULL;
       word=text;                    /*输入的所有单词要求放到 text 中，故初始
word=text */
       while ((len=getword(word))!=0)
         { root = insert(root , word); /*输入单词，若不空则将其插入到二叉排序树*/
           word + =(len+1);          /*调整下一个将要输入的单词的储存位置*/
         }
       print_tree(root);             /*打印输出*/
    }
```

本程序由 4 个部分组成，3 个子程序和一个主程序。子程序 getword 用来输入单词；子程序 insert 用来将单词插入到中序排序二叉树中；子程序 print_tree 用来中序遍历二叉排序树，并打印。单词存放在数组 text 中，且单词和单词之间用字符“0”分开。初始时数组 word 和 text 的首地址相等。

20.4 贪婪法

贪婪法是一种重要的算法设计技术，它总是做出在当前来说是最好的选择，而并不从整体上加以考虑，它所做的每步选择只是当前步骤的局部最优选择，但从整体来说不一定是最优的选择。由于它不必为了寻找最优解而穷尽所有可能解，因此其耗费时间少，一般可以快速得到满意的解。当然，我们也希望贪婪算法所得到的最终解是整体的最优解。对贪婪法的理解如下。

- 贪婪法不追求最优解，只求可行解，通俗来讲就是不求最好，只求可好。
- 每一步都按贪婪准则找一个解，故到n步后（n为问题的规模）得到问题的所有解。如找不到所有解，则修改贪婪准则，放宽贪婪条件或修改算法某些细节，重新从头开始找解。
- 每一步所找到的解是这一步中的最优解（按贪婪准则来说），但每步最优解所形成的整体解并不一定最优。

算法思想如下。

在贪婪算法中采用逐步构造最优解的方法。在每个阶段，都做出一个看上去最优的决策（在一定的贪婪标准下），决策一旦做出，就不可再更改。做出贪婪决策的依据称为贪婪准则。

应用贪婪法求解问题，首要的问题就是要弄清楚贪婪准则。但是我们应该看到，虽然在每个阶段做出的都是看上去最优的决策，但这些决策的集合从整体来说有可能并不是最优的。

例如，一个小孩买了价值少于 1 美元的糖，并将 1 美元钱交给售货员。售货员希望用数目最少的硬币找给小孩。假设提供了数目不限的面值为 25 美分、10 美分、5 美分和 1 美分的硬币。售货员分步骤组成要找的零钱数，每次加入一个硬币。

选择硬币时所采用的贪婪准则如下：每一次选择应使零钱数尽量增大。为保证解法的可行性（即所给的零钱等于要找的零钱数），所选择的硬币不应使零钱总数超过最终所需的数目。假设需要找给小孩 67 美分，首先入选的是两枚 25 美分的硬币，第三枚入选的不能是 25 美分的硬币，否则硬币的选择将不可行（零钱总数超过 67 美分），第三枚应选择 10 美分的硬币，然后是 5 美分的，最后加入两个 1 美分的硬币。可以证明采用上述贪婪算法找零钱时所用的硬币数目的确最少，这是一个得到最优解的例子。

同样是找零钱问题，如果假设提供的是数目不限的面值为 8 美分、5 美分和 1 美分的硬币。若需要找给小孩 20 美分，按照上述贪婪准则，首先入选的是两枚 8 美分硬币，然后是 4 枚 1 分硬币，共找回 6 枚硬币。显然这不是最优的解，最优解应该是 4 枚 5 美分的硬币。虽然在找零钱的每步都应用了最优找法，但后一个例子的解却不是最优解。

所以贪婪法是一种不求最优解，只是希望得到满意解的方法，即不求最好，只求可好，求第一次满足条件的解。一般来说，这个解却是最优解的很好的近似解。当然，贪婪法所求得的解也有可能是最优解，而且对范围相当广的许多问题它能产生整体最优解。下面介绍一些应用贪婪算法的典型问题。

20.4.1 背包问题

问题描述：给定 n 种物品和一个背包。物品 i 的重量是 w_i，其价值为 v_i，背包的容量为 C，问应该如何选择装入背包的物品，使得装入背包中的物品的总价值最大？

背包问题可分为如下两种。

- 0-1背包问题：对于每种物品i装入背包只有一种选择，即要么装入背包或不装入背包，不能装入多次或只装入部分。
- 部分背包问题：对于每种物品i可以只装入部分。

可以采用贪婪算法来解决背包问题。先对每种物品计算其单位重量价值 v_i/w_i，然后按单位价值单调递减的顺序对所有的物品进行排序。按照贪婪准则，开始时放入背包的物品单位重量价值尽可能大，基于这种思想，可以将按单位重量价值排序好的物品依次放入到背包中。当某个物品装入过程中，若其重量大于背包所剩余装载重量，对于 0-1 背包问题，此时装入过程完成，得到问题的解，很显然这不是问题的最优解。对于部分背包问题，可以将部分的最后物品装入背包，使得背包的重量容量被装满，显然此时背包物品的总价值要大于 0-1 背包，是问题的一个最优解。上述两个背包问题的举例如图 20-2 所示。

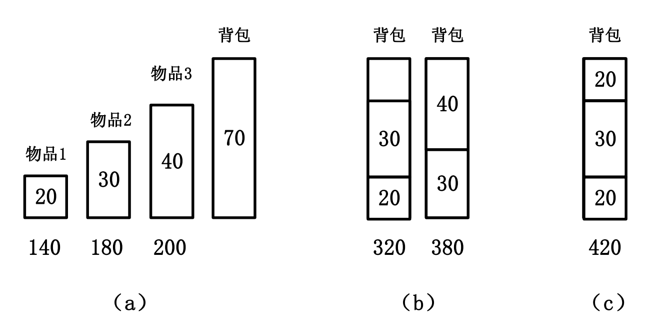


图 20-2　背包问题的一个例子

图 20-2 例子中，图 20-2（b）为 0-1 背包问题，可知由贪婪算法所得解中背包物品总价值只有 320，而最优解为 380。图 20-2（c）为部分背包问题，由贪婪算法所得解中物品总价值为 420，为最优解。

对于 0-1 背包问题，贪婪法之所以不能得到最优解是因为它无法保证最终能将背包容量装满，背包空间的闲置使得背包所装物品的总价值降低了。

0-1 背包问题的算法简单描述如下：

（1）输入物品个数 n，每个物品的重量 w_i 和价值 v_i。

（2）对物品按单位重量价值 w_i/v_i 从大到小进行排序。

（3）将排序后的物品依次装入背包。对于当前物品 i，若背包剩余可装重量大于或等于 w_i，则将物品 i 装入背包，继续考虑下一个物品 $i+1$，重复步骤（3），否则得到问题的解，输出。

0-1 背包问题的算法程序实现见程序 20-5。

【程序 20-5】

```
#include<stdio.h>
#define MAX 100                          /*最多物品数*/
 sort(int n,float a[MAX],float b[MAX])  /*对储存物品重量和价值的数组进行排序 */
{                                    /*采用冒泡法排序*/
  int j,p,h,k;
  float t1,t2,t3,c[MAX];
  for(k=1;k<=n;k++)                          /*求物品单位重量价值*/
     c[k]=a[k]/b[k];
  for(h=n;h>1;h=p){
    for(p=j=1;j<h;j++)
      if(c[j]<c[j+1]){
         t1=a[j];a[j]=a[j+1];a[j+1]=t1;
        t2=b[j];b[j]=b[j+1];b[j+1]=t2;
        t3=c[j];c[j]=c[j+1];c[j+1]=t3;
         p=j;
       }
    }
}
   /*背包装载物品子程序，limitw 为背包可装载重量*/
```

```
knapsack(int n,float limitw,float v[MAX],float w[MAX],int x[MAX])
{  float c1;                  /*c1 为背包剩余可装载重量*/
   int i;
   sort(n,v,w);               /*物品按单位重量大小降序排序*/
    c1=limitw;
    for(i=1;i<=n;i++){
      if(w[i]>c1)break;
      x[i]=1;                 /*x 储存物品选择情况，当 x[i]为 1 时，物品 i 在解中*/
      c1-=w[i];
    }
  /*对于部分背包问题，此行需添加语句 if(i<=n)x[i]=c1/w[i];*/
}
main()
{
  int n,i,x[MAX];
  float v[MAX],w[MAX],totalv=0,limitw;
  printf("please input n and limitw:");
  scanf("%d,%f",&n,&limitw);
  for(i=1;i<=n;i++)x[i]=0;      /*物品选择情况表初始化为 0*/
  for(i=1;i<=n;i++)
    {
      printf("please input %d thing's value and weight:\n",i);
      scanf("%f,%f",&v[i],&w[i]);
    }
  knapsack(n,limitw,v,w,x);
  printf("the selection is:\n");
  for(i=1;i<=n;i++)
    {
     printf("%d,",x[i]);
     totalv+=v[i]*x[i];         /*背包所装载总价值*/
    }
    printf("\n");
    printf("the total value is: %f",totalv);
}
```

贪婪法应用于 0-1 背包问题往往得不到最优解，下面是一种获得 0-1 背包问题最优解的算法。算法思想见递归法部分，同样的算法思想，可考虑非递归的程序解。为了提高找解速度，算法简单来说就是考虑每个物品对候选解的影响来形成有效的临时候选解。一个有效临时候选解是通过依次考查每个物品形成的，对物品 i 的考查有这样两种情况：当考虑该物品 i 包含在候选解中时，如果其依旧满足解的总重量的限制，则应该将该物品包含在候选解中；反之，如果不满足解的总重量限制，则该物品不应该包括在当前正在形成的候选解中。第二种是当考虑物品 i 不包含在候选解中时，且有可能找到比目前临时最佳解更好的候选解时（此时条件为期望的总价值减去当前物品 i 的价值后仍大于目前临时最佳解），则应该将该物品不包含在候选解中；反之，该物品不包括在当前候选解中的方案也不应继续考虑，则回退。对于任一值得继续考虑的方案，程序就去进一步考虑下一个物品。程序实现见程序 20-6。

【程序 20-6】

```
# include   <stdio.h>
# define    N   100
double  limitW;
int cop[N];          /*临时最佳候选解的选择方案，当 cop[i]为 1，物品 i 在解中*/
struct ele{
double  weight;
        double  value;
        } a[N];      /*储存物品重量和价值的结构*/
int k,n;
struct{ int flg;     /*物品的考虑状态，0：不选，1：将被考虑，2：曾被选中*/
        double tw;       /*背包中已经装入的总重量*/
        double tv;       /*期望达到的总价值*/
        }twv[N];         /*当前候选解中各个物品的考虑和选择状态*/
void next(int i,double tw,double tv)    /*将考虑物品 i 是否可以放入背包*/
{twv[i].flg=1; twv[i].tw=tw; twv[i].tv=tv; }
double find(struct ele *a,int n)
{   int i,k,f;
    double maxv,tw,tv,totv;
    maxv=0;
    for (totv=0.0,k=0;k<n;k++) totv+=a[k].value;   /*初始时背包期望能装载总
价值为所有物品总价值*/
    next(0,0.0,totv);                              /*0 号物品将被考虑*/
    i=0;
    While (i>=0)
    {  f=twv[i].flg; tw=twv[i].tw; tv=twv[i].tv;
       switch(f)
        {  case 1: twv[i].flg++;                      /*先考虑选中的情况*/
                if (tw+a[i].weight<=limitW)           /*选中是否满足条件*/
                 if (i<n-1)                         /*是否是最后一个物品*/
                {  next(i+1,tw+a[i].weight,tv);     /*当前物品 i 被选中，继续考虑下
                                                   一个物品*/
                        i++;
                      }
                   else {                           /*是一个更好的有效候选解*/
                        maxv=tv;
                        for (k=0;k<n;k++)
                          cop[k]=twv[k].flg!=0;
                        }
                      break;
               case 0: i--; break;                  /*回退*/
                      default:twv[i].flg=0;            /*f= =2 的情况，即被考虑选
中的物品 i 不满足重量条件*/
                      if (tv-a[i].value>maxv)          /*不选物品 i 可行吗*/
                          if (i<n-1)                /*是否是最后一个物品*/
```

```
                        {           /*当前物品i被考虑不选中，继续考虑下一个物品*/
                            next(i+1,tw,tv-a[i].value);
                             i++;
                        }
                        else {                  /*是一个更好的有效候选解*/
                             maxv=tv-a[i].value;
                             for (k=0;k<n;k++)
                                 cop[k]=twv[k].flg!=0;
                           }
                    break;
            }
        }
    return maxv;
}
void main()
{   double maxv;
    printf("输入物品种数\n"); scanf("%d",&n);
    printf("输入限制重量\n"); scanf("%lf",&limitW);
printf("输入各物品的重量和价值\n");
    for (k=0;k<n;k++)
        scanf("%lf%lf",&a[k].weight,&a[k].value);
    maxv=find(a,n);
    printf("\n选中的物品为\n");
for (k=0;k<n;k++)
        if (cop[k]) printf("%4d",k);
    printf("\n总价值为%2f\n",maxv);
}
```

20.4.2 装箱问题

问题描述：设有编号为 0，1，…，n−1 的 n 种物品，体积分别为 v_0、v_1、…、v_{n-1}。将这 n 种物品装到容量都为 V 的若干箱子里。约定这 n 种物品的体积均不超过 V，即对于 $0 \leqslant i < n$，有 $0 < v_i \leqslant V$。可知选择不同的装箱方案所需要的箱子数目可能不同。装箱问题要求使装尽这 n 种物品的箱子数要少。

要寻找该问题的最优解，可以将 n 种物品划分为小于或等于 n 的子集，考查所有的划分，可以找出满足条件且箱子数最少的划分，即为最优解。但要穷尽所有可能划分的总数则太大。对于大到一定程度的 n，找出所有可能的划分要花费的时间是无法承受的。为此，对装箱问题采用寻找最优解的近似算法，即贪婪法，可以很快地找到最优解的近似解。该算法贪婪准则是：依次将物品放到它第一个能放进去的箱子中，若当前箱子装不下当前物品，则启用一个新的箱子装该物品，直到所有的物品都装入了箱子。该算法虽不能保证找到最优解，但还是能找到非常好的解而不失一般性。设 n 件物品的体积是按从大到小排好序的，即有 $v_0 \geqslant v_1 \geqslant \cdots \geqslant v_{n-1}$。如不满足上述要求，只要先对这 n 件物品按它们的体积从大到小排序，然后按排序结果对物品重新编号即可。

算法简单描述：

```
{    输入箱子的容积;
     输入物品种数 n;
     按体积从大到小顺序，输入各物品的体积;
     预置已用箱子链为空;
     预置已用箱子计数器 box_count 为 0;
     for (i=0;i<n;i++)
     {    从已用的第一只箱子开始顺序寻找能放入物品 i 的箱子 j;
          if （已用箱子都不能再放物品 i）
          {    新启用一个箱子，并将物品 i 放入该箱子;
               box_count++;
          }
          else
               将物品 i 放入箱子 j;
     }
}
```

上述算法一次就能求出需要的箱子数 box_count，并能求出各箱子所装物品，但该算法不一定能找到最优解。如下面例子所示，设有 6 种物品，它们的体积分别为：60、45、35、20、20 和 20 单位体积，箱子的容积为 100 个单位体积。按上述算法计算，需 3 只箱子，各箱子所装物品分别为：第一只箱子装物品 1、3；第二只箱子装物品 2、4、5；第三只箱子装物品 6。而最优解为两只箱子，分别装物品 1、4、5 和 2、3、6。该算法也能够找到最优解。下面的例子说明了这种情况，设有 6 种物品，它们的体积分别为：60、35、25、20、20 和 20 单位体积，箱子的容积为 100 个单位体积。按上述算法计算，可以得到最优解为两个箱子，分别装物品 1、2 和 3、4、5、6。

该问题算法程序实现见程序 20-7。

【程序 20-7】

```
# include   <stdio.h>
# include   <stdlib.h>
typedef  struct  ele{           /*物品结构的信息*/
    int  vno;                   /*物品号*/
    struct  ele  *link;         /*指向下一物品的指针*/
}ELE;
typedef  struct  hnode{         /*箱子结构信息*/
int  remainder;                 /*箱子的剩余空间*/
    ELE  *head;             /*箱子内物品链的首元指针*/
    struct  hnode  *next;       /*箱子链的后继箱子指针*/
}  HNODE;
void  main()
{   int  n, i, box_count, box_volume, *a;
    HNODE  *box_h,  *box_t,  *j;
    ELE   *p,  *q;
    printf("输入箱子容积\n"); scanf("%d",&box_volume);
    printf("输入物品种数\n"); scanf("%d",&n);
```

```
    a=(int *)malloc(sizeof(int)*n);
    printf("请按体积从大到小顺序输入各物品的体积：");
    for (i=0;i<n;i++)scanf("%d",a+i);           /*数组 a 按从大到小顺序存放各
                                                  物品的体积信息*/
    box_h=box_t=NULL;                   /*box_h 为箱子链的首元指针，box_t 为当前箱
                                          子的指针，初始为空* /
    box_count=0;              /*箱子计数器初始也为 0*/
    for (i=0;i<n;i++)             /*物品 i 按下面各步开始装箱*/
    {   p=(ELE *)malloc(sizeof(ELE));
        p->vno=i;                 /*指针 p 指向当前待装物品*/
        /*从第一只箱子开始顺序寻找能放入物品 i 的箱子 j*/
        for (j=box_h;j!=NULL;j=j->next)
            if (j->remainder>=a[i])break;       /*找到可以装物品 i 的箱子，贪婪
                                                  准则的体现*/
           if (j= =NULL) {            /*已使用的箱子都不能装下当前物品 i*/
               j=(HNODE *)malloc(sizeof(HNODE));   /*启用新箱子*/
               j->remainder=box_volume-a[i];   /*将物品 i 放入新箱子 j*/
               j->head=NULL;              /*新箱子内物品链首元指针初始为空*/
               if (box_h= =NULL) box_h=box_t=j;     /*新箱子为第一个箱子*/
               else box_t=boix_t->next=j;   /*新箱子不是第一个箱子*/
               j->next=NULL;
              box_count++;
          }
          else  j->remainder一=a[i];         /*将物品 i 放入已用过的箱子 j*/
          /*物品放入箱子后要修改物品指针链*/
          for (q=j->head;q!=NULL&&q->link!=NULL;q=q->link);
              if (q= =NULL) {                 /*新启用的箱子插入物品*/
                p->link=j->head; j->head=p;     /*p 为指向当前物品的指针*/
               }
             else{                          /*已使用过的箱子插入物品*/
               p->link=NULL; q->link=p;      /*q 为指向箱子内物品链顶端的物品*/
             }
    }
    printf("共使用了%d 只箱子", box_count);
    printf("各箱子装物品情况如下：");
    for (j=box_h,i=1;j!=NULL;j=j->next,i++)         /*输出 i 只箱子的情况*/
    {   printf("第%2d 只箱子，还剩余容积%4d，所装物品有;\n",i,j->remainder);
        for (p=j->head;p!=NULL;p=p->link)
            printf("%4d",p->vno+1);
        printf("\n");
    }
}
```

装箱问题所采用的数据结构为链表。用链表将启用的箱子链接起来，而且每个箱子所装入的物品也用一个链表将它们链接起来，这样就有两个链表：箱子链和物品链。程序将

物品按体积从大到小依次装入箱子。对每一个物品，从箱子链中第一个箱子开始顺序寻找能放入该物品的箱子（箱子剩余体积大于或等于当前物品体积），将物品放入最先找到的箱子。装箱的贪婪准则是：如果一旦找到能装下当前物品的箱子，就将当前物品放入，而不考虑其最优解情况；若所有启用的箱子都装不下当前物品，则开启一个新箱子。

程序中箱子链首元指针为 box-h，当前箱子指针为 box-t，箱内物品链首元指针为 j->head。装入物品 5 的过程如图 20-3 所示。

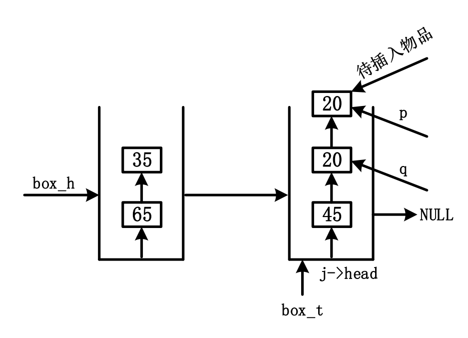


图 20-3　物品装入过程图

20.4.3　哈夫曼编码问题

问题描述：在数据通信中，一般需要将传送的文字转换成由二进制字符 0、1 组成的二进制串，称为编码。例如，假设要传送的电文为 AABCADC，电文中只含有 A、B、C、D4 种字符，若这 4 种字符采用表 20-1（a）所示的编码，则电文的代码为 000000010100000111100，长度为 21。在传送电文时，我们总是希望传送时间尽可能短，这就要求电文代码尽可能短，显然，这种编码方案产生的电文代码不够短。表 20-1（b）所示为另一种编码方案，用此编码对上述电文进行编码所建立的代码为 00000110001110，长度为 14。在这种编码方案中，4 种字符的编码均为两位，是一种等长编码。如果在编码时考虑字符出现的频率，让出现频率高的字符采用尽可能短的编码，出现频率低的字符采用稍长的编码，构造一种不等长编码，则电文的代码就可能更短。如当字符 A、B、C、D 采用表 20-1（c）所示的编码时，上述电文的代码为 0011010011110，长度仅为 13。构造一种编码方案，使得电文的编码总长度最短。

表 20-1　字符的 4 种不同的编码方案

字符	编码
A	000
B	010
C	100
D	111

（a）

字符	编码
A	00
B	01
C	10
D	11

（b）

字符	编码
A	0
B	110
C	10
D	111

（c）

字符	编码
A	01
B	010
C	001
D	10

（d）

采用哈夫曼编码方案，即应用哈夫曼树构造使电文的编码总长最短的编码方案。假设有 n 个权值｛w_1，w_2，…，w_n｝，构造有 n 个叶子结点的二叉树，每个叶子结点带权为 w_i，

则二叉树带权路径长度 WPL 为树中所有叶子结点到树根之间的路径长度与结点上权的乘积之和，WPL 最小的二叉树为哈夫曼树。应用哈夫曼树编码的方法如下：设需要编码的字符集合为$\{d_1, d_2, \cdots, d_n\}$，它们在电文中出现的次数集合相应为$\{w_1, w_2, \cdots, w_n\}$，以$d_1$，$d_2$，…，$d_n$ 作为叶结点，w_1，w_2，…，w_n 作为它们的权值，构造一棵哈夫曼树，规定哈夫曼树中的左分支代表 0，右分支代表 1，则从根结点到每个叶结点所经过的路径分支组成的 0 和 1 的序列便为该结点对应字符的编码，称为哈夫曼编码。

在哈夫曼编码树中，树的带权路径长度 WPL 含义是各个字符的码长与其出现次数的乘积之和，也就是电文的代码总长，所以采用哈夫曼树构造的编码是一种能使电文代码总长最短的不等长编码。

此外，在建立不等长编码时，必须使任何一个字符的编码都不是另一个字符编码的前缀，这样才能保证译码的唯一性。例如，表 20-1（d）的编码方案，字符 A 的编码 01 是字符 B 的编码 010 的前缀部分，这样对于代码串 0101001，既是 AAC 的代码，也是 ABD 和 BDA 的代码，因此，这样的编码不能保证译码的唯一性，称为具有二义性的译码。

然而，采用哈夫曼树进行编码，则不会产生上述二义性问题。因为，在哈夫曼树中，每个字符结点都是叶结点，它们不可能在根结点到其他字符结点的路径上，所以一个字符的哈夫曼编码不可能是另一个字符的哈夫曼编码的前缀，从而保证了译码的非二义性。

算法简单描述如下。

这是一种构造最优无前缀码的贪婪算法，用于求解某个字符串的哈夫曼编码，分为以下两步。

1．构造哈夫曼树

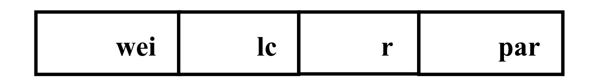
构造哈夫曼树的算法为哈夫曼算法，其过程如下。

①根据给定的 n 个权值 $\{w_1, w_2, \cdots, w_n\}$ 构造含 n 棵二叉树的集合 F= $\{T_1, T_2, \cdots, T_n\}$，其中，$T_i$ 只有一个带权为 w_i 的根结点，其左右子树为空。

②以 T_i 为子树逐步合并形成哈夫曼树。根据贪婪准则，在 F 中选取两棵根结点权值最小的树作为左右子树形成一个新二叉树，且新二叉树根结点权值为其左右子树根结点的权值之和。同时在 F 中用新二叉树替代它的左右子树。

③重复上述步骤，直到 F 只含有一棵树为止。这棵树即为哈夫曼树。

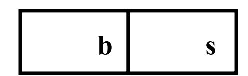
可以设置一个结构数组 HuffNode 保存哈夫曼树中各结点的信息。根据二叉树的性质可知，具有 n 个叶子结点的哈夫曼树共有 $2n-1$ 个结点，所以数组 HuffNode 的大小设置为 $2n-1$，数组元素的结构形式如下：



其中，weight 域保存结点的权值，lchild 和 rchild 域分别保存该结点的左、右孩子结点在数组 HuffNode 中的序号，从而建立起结点之间的关系。为了判定一个结点是否已加入到要建立的哈夫曼树中，可通过 parent 域的值来确定。初始时 parent 的值为−1，当结点加入到树中时，该结点 parent 的值为其双亲结点在数组 HuffNode 中的序号，就不会是−1。

2．在哈夫曼树上求叶结点的编码

该过程实质上就是在已建立的哈夫曼树中，从叶结点开始，沿结点的双亲链域回退到根结点，每回退一步，就走过了哈夫曼树的一个分支，从而得到一位哈夫曼码值。由于一个字符的哈夫曼编码是从根结点到相应叶结点所经过的路径上各分支所组成的 0、1 序列，因此先得到的分支代码为所求编码的低位码，后得到的分支代码为所求编码的高位码。可以设置一个结构数组 HuffCode 用来存放各字符的哈夫曼编码信息，数组元素的结构如下：



其中，分量 bit 为一维数组，用来保存字符的哈夫曼编码，start 表示该编码在数组 bit 中的开始位置。所以，对于第 i 个字符，它的哈夫曼编码存放在 HuffCode[i].bit 中的从 HuffCode[i].start 到 n 的分量上。

该问题算法程序实现见程序 20-8。

【程序 20-8】

```
#define MAXBIT 10                 /*定义哈夫曼编码的最大长度*/
#define MAXVALUE 10000            /*定义最大权值*/
#define MAXLEAF 30               /*定义哈夫曼树中最多叶子结点个数*/
#define MAXNODE  MAXLEAF*2-1     /*哈夫曼树最多结点数*/
typedef struct {                    /*哈夫曼编码信息的结构*/
    int bit[MAXBIT];
    int start;
    }HCodeType;
typedef struct {                   /*哈夫曼树结点的结构*/
    int weight;
    int parent;
    int lchild;
    int rchild;
    }HNodeType;
void HuffmanTree(HNodeType HuffNode[MAXNODE],int n)/*构造哈夫曼树的函数*/
{
  int i,j,m1,m2,x1,x2;
  for(i=0;i<2*n-1;i++)         /*存放哈夫曼树结点的数组 HuffNode[ ]初始化*/
  {    HuffNode[i].weight=0;
       HuffNode[i].parent=-1;
      HuffNode[i].lchild=-1;
      HuffNode[i].rchild=-1;
  }
  for(i=0;i<n;i++)            /*输入 n 个叶子结点的权值*/
  {
  printf("please input %d character's weight\n",i);
  scanf("%d",&HuffNode[i].weight);
  }
  for(i=0;i<n-1;i++)          /*该循环开始构造哈夫曼树*/
    {  m1=m2=MAXVALUE;
```

```
        x1=x2=0;
         for(j=0;j<n+i;j++)
         { if(HuffNode[j].weight<m1&&HuffNode[j].parent= =-1)
             { m2=m1; x2=x1;m1=HuffNode[j].weight;  x1=j;}
           else if(HuffNode[j].weight<m2&&HuffNode[j].parent= =-1)
             { m2=HuffNode[j].weight; x2=j; }
          }
        HuffNode[x1].parent=n+i;    /*以下代码将找出的两棵子树合并为一棵子树*/
        HuffNode[x2].parent=n+i;
        HuffNode[n+i].weight=HuffNode[x1].weight+HuffNode[x2].weight;
        HuffNode[n+i].lchild=x1;
        HuffNode[n+i].rchild=x2;
      }
}
  void main()
 {
   HNodeType HuffNode[MAXNODE];
   HCodeType HuffCode[MAXLEAF],cd;
   int i,j,c,p,n;
   printf("please input n:\n");
   scanf("%d",&n);                /*输入叶子结点个数*/
   HuffmanTree(HuffNode,n);         /*建立哈夫曼树 */
   for(i=0;i<n;i++)       /*该循环求每个叶子结点对应字符的哈夫曼编码*/
    {  cd.start=n-1;   c=i;
      p=HuffNode[c].parent;
       while(p!=-1)                /*由叶结点向上直到树根*/
         { if(HuffNode[p].lchild= =c)cd.bit[cd.start]=0;
          else  cd.bit[cd.start]=1;
          cd.start--;  c=p;
          p=HuffNode[c].parent;
          }
        /*保存求出的每个叶结点的哈夫曼编码和编码的起始位*/
        for (j=cd.start+1;j<n;j++)
          HuffCode[i].bit[j]=cd.bit[j];
         HuffCode[i].start=cd.start;
      }
   for(i=0;i<n;i++)                 /*输出每个叶子结点的哈夫曼编码*/
  {   printf("%d character is: ",i);
        for(j=HuffCode[i].start+1;j<n;j++)
            printf("%d",HuffCode[i].bit[j]);
         printf("\n");
    }
 }
```

构造哈夫曼树时，首先将由 n 个字符形成的 n 个叶结点存放到数组 HuffNode 的前 n 个分量中，然后根据前面介绍的哈夫曼方法的基本思想，不断将两棵小子树合并为一个较大的子树，每次构成的新子树的根结点顺序放到 HuffNode 数组中的前 n 个分量的后面。电

文 AABCADC 中字符 A、B、C、D 的哈夫曼编码过程如图 20-4 所示，图中结点圆内的数字为结点权值，叶结点权值为相应字符在电文中出现的次数。

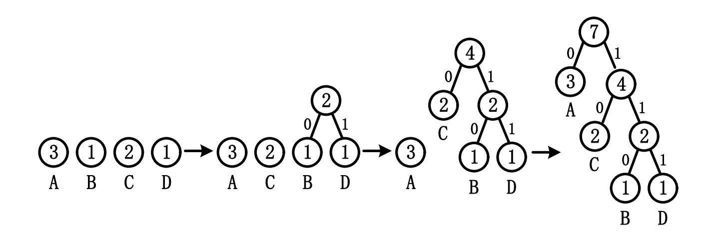


图 20-4　哈夫曼编码过程图

20.5　回溯法

回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但当搜索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择。这种走不通就退回再走的技术就是回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

可用回溯法求解的问题 P，通常要能表达为：对于已知的由 n 元组（x_1，x_2，…，x_n）组成的一个解空间 E={（x_1，x_2，…，x_n）｜$x_i \in S$ ，i=1，2，…，n}，给定关于 n 元组中分量的一个约束集 D，问题 P 需要求出 E 中满足 D 的所有 n 元组，其中 S 是分量 x_i 的定义域，且 $|S|$ 有限，i=1，2，…，n。称 E 中满足 D 的任一 n 元组为问题 P 的一个解。

解问题 P 的最朴素的方法就是穷举法，即对 E 中的所有 n 元组逐一地检测其是否满足 D 的全部约束，若满足，则为问题 P 的一个解，但显然，其计算量是相当大的。

可以发现，对于许多问题，只要存在 $0 \leqslant j \leqslant n-1$，使得（$x_1$，$x_2$，…，$x_j$）违反 D 的约束，则以（x_1，x_2，…，x_j）为前缀的任何 n 元组（x_1，x_2，…，x_j，x_{j+1}，…，x_n）一定也违反 D 的约束，$n \geqslant i > j$。因此，可以肯定，一旦检测断定某个 j 元组（x_1，x_2，…，x_j）违反 D 的约束，就可以肯定，以（x_1，x_2，…，x_j）为前缀的任何 n 元组（x_1，x_2，…，x_j，x_{j+1}，…，x_n）都不会是问题 P 的解，因而就不必去搜索它们、检测它们。回溯法正是针对这类问题，利用这类问题的上述性质而提出来的比穷举法效率更高的算法。

回溯法首先将问题 P 的 n 元组的解空间 E 表示成一棵高为 n 的带权有序树 T（称为解空间树），把在 E 中求问题 P 的所有解转化为在 T 中搜索问题 P 的所有解。例 1 说明了 T 的建立和利用 T 求解的过程。

【例 1】n=5，r=3 的所有组合为：

（1）1、2、3　　（6）1、4、5

（2）1、2、4　　（7）2、3、4

（3）1、2、5　　（8）2、3、5

（4）1、3、4　　（9）2、4、5

（5）1、3、5　　（10）3、4、5

则该问题的解空间为：

E=｛（X_1，X_2，X_3）｜$X_i \in$S，i=1，2，3 ｝，其中 X_i 的定义域为：S={1，2，3，4，5}，约束集 D 为：$X_1<X_2<X_3$。则建立的问题解空间树 T 如图 20-5 所示。

如图 20-5 所示，组合问题解空间树的每层路径表示解空间｛(X_1、X_2、X_3)｝的一个分量，路径的权表示该分量的所有取值。求解从 T 的根结点出发，按深度优先的策略，系统地搜索以该结点为根的子树中可能包含着解的所有状态结点，而跳过对肯定不含解的所有子树的搜索，以提高搜索效率。在组合问题中，从 T 的根出发深度优先遍历该树。当遍历到结点（1，1）时，它不满足约束条件 D，则遍历跳过该结点的所有子树，回溯至该结点的父结点，遍历该父结点的下一个子树；当遍历到结点（1，2）时，虽然它满足约束条件，但还不是解结点，则应继续深度遍历；当遍历到叶子结点（1，2，1）时，它不满足约束条件 D，则同样需要回溯；当遍历到叶子结点（1，2，3）时，由于它已是一个解结点，则保存（或输出）该结点，并需要回溯到其父结点，继续深度遍历该父结点的下一个子树；当遍历到结点（1，5）时，由于它不是叶子结点，但不满足约束条件，故也需回溯。按同样方法依次遍历完整棵解空间树，就能找到所有的解结点，输出。

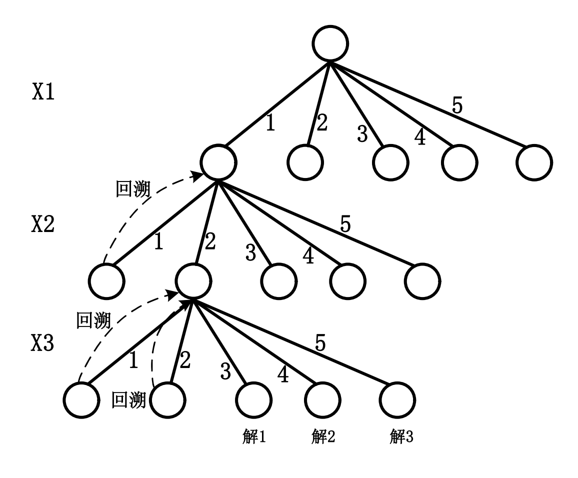


图 20-5　组合问题的状态空间树 T 示意图

由上可以看出，回溯法首先放弃关于规模大小的限制，并将问题的候选解按某种顺序逐一试探，故其又称为试探法。试探过程中发现当前候选解不可能是解，且该规模下还有其他可选候选解时，就顺序试探下一个候选解；试探过程中发现当前候选解不可能是解，且该规模下没有其他可选候选解时，就缩小规模，试探该规模的下一个候选解；如果当前候选解除不满足问题规模之外，满足其他要求，则扩大规模，继续试探；如果当前候选解满足包括问题规模在内的所有要求时，则该候选解就是问题的一个解。

下面介绍一些应用回溯法的典型问题。

20.5.1　组合问题

问题描述：找出从自然数 1，2，…，n 中任取 r 个数的所有组合。

采用回溯法找问题的解，将找到的组合以从小到大的顺序存于 a[0]，a[1]，…，a[r−1]中，组合的元素满足如下性质：

- a[i+1]>a[i]，后一个数字比前一个大。
- a[i]−i<=n−r+1。

算法简单描述如下。

按回溯法的思想，由如下步骤得到问题的解。

（1）首先设置 a[0]=1，此时候选解的规模为 1，候选解为 1，且满足除问题规模之外的全部条件，则下一步应扩大规模，考虑 a[1]的赋值。

（2）设置 a[1]= a[0]＋1=2，此时候选解规模为 2，候选解为 1、2，且仍不满足问题的规模，则继续扩大规模；设置 a[2]= a[1]＋1=3，此时候选解规模为 3，候选解为 1、2、3，该候选解满足包括问题规模在内的全部条件，因而是一个解，输出。

（3）再考虑该规模 3 下是否还有其他解。选下一个候选解，因此令 a[2]加 1 调整为 4，以及以后再加 1 调整为 5 都满足问题的全部要求，得到解 1、2、4 和 1、2、5，输出。

（4）规模 3 情况下的候选解已经考查完，下一步应该回溯考虑规模 2 情况下的下一个候选解。则令 a[1]加 1 为 3，候选解为 1、3，此时规模不满足，继续扩大规模，设置 a[2]= a[1]＋1=4，此时候选解规模为 3，候选解为 1、3、4，该候选解满足包括问题规模在内的全部条件，因此为一个解，输出。重复上述向前试探和向后回溯，直至要从 a[0]再回溯时，说明已经找完问题的全部解。由于数组 a 的元素始终按递增顺序增加，故其始终满足上述第一个条件。

该问题算法实现见程序 20-9。

【程序 20-9】

```
# define    MAXN    100
int a[MAXN];
void comb(int m,int r)          /*求从自然数 1 到 m 中任取 3 个数的所有组合子程序*/
{   int i,j;
    i=0; a[i]=1;            /*初始规模为 1 时，a[0]为 1*/
    do {
        if (a[i]-i<=m-r+1)  /*还可以向前试探*/
        {   if (i= =r-1)    /*当前候选解的规模满足问题的规模要求，找到一个解*/
            {   for (j=0;j<r;j++)
                    printf("%4d",a[j]);
                printf("\n");
            }
            a[i]++;     /*考查当前规模的下一个候选解*/
            continue;
            }
            i++;
            a[i]=a[i-1]+1;
        }
        else            /*当前规模的候选解已经全部考查完，则应回溯，缩小规模*/
        {   if (i= =0)       /*回溯至初始规模，则已经全部找到了解*/
                return;
            a[--i]++;   /*缩小规模，考查下一个候选解*/
        }
    }while (1)
}

main()
{comb(5,3); }
```

20.5.2　子集和问题

问题描述：给定由 n 个不同正数组成的集合 $W=\{w_1, w_2, \cdots, w_n\}$ 和正数 M，要求找出 $N=\{1, 2, \cdots, n\}$ 的所有使得 $\sum_{i\in s} w_i = M$ 的子集 S。例如，给定 $n=4$，$W=\{11, 13, 24, 7\}$ 和 $M=31$，则相应的子集和问题的解是 $\{3, 4\}$ 和 $\{1, 2, 4\}$。

这个问题还可表述为：求所有使得 $\sum_{i\in s} w_i x_i = M$ 的 n 元组（x_1，x_2，$\cdots$，x_n），其中 $x_i\in\{0, 1\}$，$1\leqslant i\leqslant n$，以及得到的 n 元组相对应的原问题的解 $S=\{i|x_i=1, i\in\{1, 2, \cdots, n\}\}$。解向量的元素 x_i 或者为 0 或者为 1，这取决于子集中是否包含 w_i。

根据题意，可知，若条件

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geqslant M$$

不成立，则（x_1，x_2，$\cdots$，x_k）就不可能成为解的一部分。此外，如果假定 w_i 按升序排列，如果条件

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leqslant M$$

不成立，则（x_1，x_2，$\cdots$，x_k）也不可能成为解的一部分。故要想使得（x_1，x_2，$\cdots$，x_k）有可能成为解的一部分，则必须满足如下两个条件：

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geqslant M \quad \text{和} \quad \sum_{i=1}^{k} w_i x_i + w_{k+1} \leqslant M$$

由上述可以看到，子集和问题的解空间树 T 是一棵高度为 n 的二叉树，其中深度为 k 的一个状态结点对应于一个 k 元组（x_1，x_2，$\cdots$，x_k）。可以约定（x_1，x_2，$\cdots$，x_{k-1}，1）和（x_1，x_2，$\cdots$，x_{k-1}，0）所对应的状态结点分别是（x_1，x_2，$\cdots$，x_{k-1}）所对应的状态结点的左儿子和右儿子。初始时，要求输入 W 时，从小到大输入，使得数组 W 中的元素有序。

该问题算法实现见程序 20-10。

【程序 20-10】

```
#include <stdio.h>
#define MAX 100
int w[MAX],x[MAX],m,n;
void sumofsub(float s,int k,float r)
{                                /*求子集和函数，s 和 r 表示如程序附后说明*/
  int i;
  x[k]=1;                        /*试探 x[k]包含在解向量中的情况*/
  if(s+w[k]= =m)                     /*找到一个解，输出*/
  {
    for(i=1;i<=k;i++) printf("%4d",x[i]);
    printf("\n");
  }
  else{
```

```
      if(s+w[k]+w[k+1]<=m)          /* x[k]包含在解向量中是否可行 */
         sumofsub(s+w[k],k+1,r-w[k]);   /* 可行则扩大规模*/
      if(s+r-w[k]>=m&&s+w[k+1]<=m){    /*试探 x[k]不包含在解向量中的情况是否可行 */
         x[k]=0;                     /* 可行则扩大规模*/
        sumofsub(s,k+1,r-w[k]);         /* 可行则扩大规模*/
      }
    }
   }
   void main()
   { int i,k;
     float r,s;
     r=s=0;
     k=1;
     for(i=0;i<=MAX;i++)x[i]=0;
     scanf("%4d,%4d",&n,&m);
     printf("please input values of array w by ascend:\n");
     for(i=1;i<=n;i++) scanf("%4d",&w[i]);     /* 按升序输入 w 数组的值*/
     for(i=1;i<=n;i++)
       r+=w[i];                             /* 计算 r 的初始值，为 w 数组所有元素的和*/
     sumofsub(s,k,r);                       /* 递归求所有全部解*/
   }
```

程序中$\sum_{i=1}^{k} w_i x_i$和$\sum_{i=k+1}^{n} w_i x_i$分别保存在变量 s 和 r 中。该算法没有明显地使用测试条件 $k>n$ 去终止递归，其原因在于过程每次调用开始时 $s\neq$M，$s+r\geqslant$M，因此，r$\neq$0，从而 k 也不可能大于 n。而且如果 $s+w[k]$=M，则 $x[k+1]$，$x[k+2]$，…，$x[n]$应该为 0，这些 0 不包含在解的输出中。而且，程序假定 $x[1]\leqslant$M，$w[1]+x[2]+\cdots+x[k]\geqslant$M。图 20-6 是 n=4，M=31，W={7,11,13,24}时的解空间树。从树中可以看到，该问题的解为（1,1,1）和（1,0,0,1）。

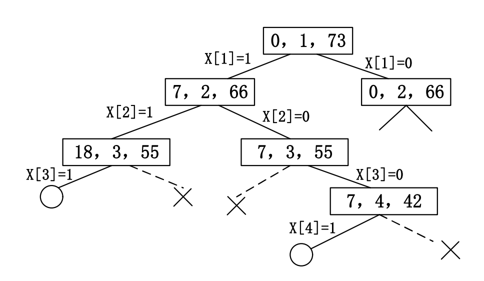


图 20-6　子集和问题解空间树

20.6　分治法

对于一个规模为 n 的问题，若该问题可以容易地解决（比如规模 n 较小）则直接解决；否则将其分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。这种算法设计策略叫作分治法。

我们知道，任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。例如，对于 n 个元素的排序

问题，当 n=1 时，不需任何计算。n=2 时，只要作一次比较即可排好序。n=3 时只要做 3 次比较即可……而当 n 较大时，问题就不那么容易处理了。要想直接解决一个规模较大的问题，有时相当困难。而分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。如果原问题可分割成 k 个子问题，$1<k\leq n$，且这些子问题都可解，并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。因此，分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

分治法所能解决的问题一般具有如下几个特征：

- 该问题的规模缩小到一定的程度就可以容易地解决。
- 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。
- 利用该问题分解出的子问题的解可以合并为该问题的解。
- 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

上述的第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般随着问题规模的增加而增加。第二条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了递归思想的应用。第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑贪心法或动态规划法。第四条特征涉及分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题。

分治法在每一层递归上都有 3 个步骤。

① 分解：将原问题分解为若干个规模较小、相互独立、与原问题形式相同的子问题。

② 解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题。

③ 合并：将各个子问题的解合并为原问题的解。

下面将分析一个分治法中的经典问题——二分法查找。

在对线性表的操作中，经常需要查找某一个元素在线性表中的位置。此问题的输入是待查元素 x 和线性表 L，输出为 x 在 L 中的位置或者 x 不在 L 中的信息。

比较自然的想法是一个一个地扫描 L 的所有元素，直到找到 x 为止。这种方法对于有 n 个元素的线性表在最坏情况下需要 n 次比较。一般来说，如果没有其他的附加信息，在有 n 个元素的线性表中查找一个元素在最坏情况下都需要 n 次比较。

下面考虑一种简单的情况。假设该线性表已经排好序了，不妨设它按照主键的递增顺序排列（即由小到大排列）。在这种情况下，是否有改进查找效率的可能呢？如果线性表里只有一个元素，则只要比较这个元素和 x 就可以确定 x 是否在线性表中。因此这个问题满足分治法的第一个适用条件。同时对于排好序的线性表 L 有如下性质：比较 x 和 L 中任意一个元素 L[i]，若 x=L[i]，则 x 在 L 中的位置就是 i；如果 x<L[i]，由于 L 是递增排序的，因此假如 x 在 L 中，x 必然排在 L[i]的前面，所以只要在 L[i]的前面查找 x 即可；如果 x>L[i]，同理只要在 L[i]的后面查找 x 即可。无论是在 L[i]的前面还是后面查找 x，其方法都和在 L 中查找 x 一样，只不过是线性表的规模缩小了。这就说明了此问题满足分治法的第二个和第三个适用条件。很显然此问题分解出的子问题相互独立，即在 L[i]的前面或后面查找 x 是独立的子问题，因此满足分治法的第四个适用条件。

于是得到利用分治法在有序表中查找元素的算法。其程序片断见程序 20-11。

【程序 20-11】

```
function Binary_Search(L,a,b,x);
 {  if (a>b) return(-1);
    else
      { m=(a+b)/2;
        if (x= =L[m]) return(m);
        else if(x>L[m])
          return(Binary_Search(L,m+1,b,x));          /*递归实现*/
        else return(Binary_Search(L,a,m-1,x));       /*递归实现*/
      }
}
```

在上述算法中，L 为排好序的线性表，x 为需要查找的元素，b、a 分别为 x 的位置的上下界，即如果 x 在 L 中，则 x 在 L[a..b]中。每次用 L 中间的元素 L[m]与 x 比较，从而确定 x 的位置范围，然后递归地缩小 x 的范围，直到找到 x。

20.7 动态规划法

动态规划是运筹学的一个分支，是求解决策过程最优化的数学方法。20 世纪 50 年代初美国数学家 R.E.Bellman 等人在研究多阶段决策过程的优化问题时，提出了著名的最优化原理，把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，创立了解决这类过程优化问题的新方法——动态规划。

在软件设计师考试中，动态规划法的程序题是最难考题之一。动态规划法与分治法有类似之处，即他们都将大的问题拆分为子问题以降低问题的复杂度，两者区别在于分治法的子问题往往是独立的，而动态规划法的子问题不是相互独立的。在考试时，要区别动态规划法与分治法，有一种简单有效的方法，即分析程序中有没有把子问题的解使用临时数组存下来，然后大的问题通过查找子问题结果表来构成，如果有这种特征，则为动态规划法。下面以多个矩阵相乘（链乘）的计算任务实例说明动态规划法的基本思想。

某工程计算中要完成多个矩阵相乘（链乘）的计算任务。

两个矩阵相乘要求第一个矩阵的列数等于第二个矩阵的行数，计算量主要由进行乘法运算的次数决定。采用标准的矩阵相乘算法，计算 $A_{m*n}*B_{n*p}$，需要 m*n*p 次乘法运算。

矩阵相乘满足结合律，多个矩阵相乘，不同的计算顺序会产生不同的计算量。以矩阵 $A1_{10*100}$，$A2_{100*5}$，$A3_{5*50}$ 三个矩阵相乘为例，若按（A_1*A_2）$*A_3$ 计算，则需要进行 10*100*5+10*5*50=7 500 次乘法运算；若按 A_1*（A_2*A_3）计算，则需要进行 100*5*50+10*100*50=75 000 次乘法运算。可见不同的计算顺序对计算量有很大的影响。

矩阵链乘问题可描述为：给定 n 个矩阵<A_1，A_2，…，A_n>，矩阵 Ai 的维数为 $p_{i-1}*P_i$，其中 i=1，2，…，n。确定一种乘法顺序，使得这 n 个矩阵相乘时进行乘法的运算次数最少。

由于可能的计算顺序数量非常庞大，对较大的 n，用蛮力法确定计算顺序是不实际的。经过对问题进行分析，发现矩阵链乘问题具有最优子结构，即若 $A_1*A_2*\cdots*A_n$ 的一个最优计算顺序从第 k 个矩阵处断开，即分为 $A_1*A_2*\cdots A_k$ 和 $A_{k+1}*A_{k+2}*\cdots*A_n$ 两个子问题，则该

最优解应该包含 $A_1*A_2*\cdots*A_k$ 的一个最优计算顺序和 $A_{k+1}*A_{k+2}*\cdots A_n$ 的一个最优计算顺序。据此构造递归式，

$$cost[i][j]=\begin{cases}0 & \text{if } i=j\\ \min_{i\leqslant k<j} cost[i][k]+cost[k+1][j]+p_i*p_{k+1}*p_{j+1} & \text{if } i<j\end{cases}$$

其中，cost[i][j]表示 $A_{i+1}*A_{i+2}*\cdots A_{j+1}$ 的最优计算的计算代价。最终需要求解 cost[0][n-1]。

程序代码算法实现采用自底向上的计算过程。首先计算两个矩阵相乘的计算量，然后依次计算 3 个矩阵、4 个矩阵、……、n 个矩阵相乘的最小计算量及最优计算顺序。程序中要用的主要变量如下：

n：矩阵数。

seq[]：矩阵维数序列。

cost[][]：二维数组，长度为 $n*n$，其中元素 cost[i][j]表示 $A_{i+1}*A_{i+2}*\cdots A_{j+1}$ 的最优计算的计算代价。

trace[][]：二维数组，长度为 $n*n$，其中元素 trace[i][j]表示 $A_{i+1}*A_{i+2}*\cdots A_{j+1}$ 的最优计算对应的划分位置，即 k。

【程序 20-12】

```
#define  N  100
int cost[N][N];
int trace[N][N];
int  cmm(int n,int seq[]){
int tempCost;
int tempTrace;
int i,j,k,p;
int temp;
for( i=0;i<n;i++){ cost[i][i] = 0;}
for(p=1;p<n;p++){
for(i=0; i<n-p ;i++){
   j=i+p;
tempCost = -1;
for(k = i;k<j;k++){
  temp = cost[i][k]+cost[k+1][j]+seq[i]*seq[k+1]*seq[j+1];
      if(tempCost==-1||tempCost>temp){
             tempCost = temp;
tempTrace=k;
}
}
cost[i][j] = tempCost;
trace[i][j] = tempTrace;
}
}
return cost[0][n-1];
}
```

在编写上述程序段时，最大的难点在于把算法中已给出的递归式转化为程序代码。在考试时，一般都已经给出递归式，所以要根据程序中变量的意思，将算法中的递归式进行精确的转换。

举报电话：（010）88254396；（010）88258888

传　　真：（010）88254397

E-mail：dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮　　编：100036